1991

Testing of a grid-based FFT n-body code for galactic simulations

Mark A. Mattox
San Jose State University

Order Number 1345810

Testing of a grid-based FFT n-body code for galactic simulations

Mattox, Mark Alan, M.S.

San Jose State University, 1991

U·M·I

300 N. Zeeb Rd.
Ann Arbor, MI 48106

TESTING OF A GRID-BASED FFT N-BODY CODE FOR GALACTIC SIMULATIONS

A Thesis

Presented to

The Faculty of the Department of Physics

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Mark A. Mattox

December, 1991

APPROVED FOR THE DEPARTMENT OF PHYSICS

Dr. Patrick Hamill

Dr. Tom Steiman-Cameron

Dr. Leslie Tomley

APPROVED FOR THE UNIVERSITY

ABSTRACT

Testing of a Grid-Based FFT N-Body Code
for Galactic Simulations

by Mark A. Mattox

Validation techniques for a 2-D Cartesian Fourier grid n-body code are described. The computer code is based on the self-gravitating n-body algorithm by Miller and Prendergast [1]. Particles are advanced by a half-step leapfrog algorithm, using a softened potential. Four tests used in the validation exercise are described. The first is a single particle potential test. The second is a central mass, 2 body problem. The third is a many body test using 10 particles. The fourth is a disk test of over 100,000 particles, where we compare dynamical evolution of the model with previously published N-body results [2], [3].

---

[1] R. H. Miller and K. H. Prendergast, Ap. J. **151**, 699 (1968).
[2] F. Hohl, J. Computational Phys. 9, **10** (1972).
[3] R. H. Miller, J. Computational Phys. **21**, 400 (1976).

This Work is Dedicated
to the Memory of My Father,
Arthur L. Mattox.

TABLE OF CONTENTS

TABLES

FIGURES

ACKNOWLEDGMENTS

This thesis could not have been accomplished without the direction and support of my committee members. I am indebted to my advisor Dr. Patrick Hamill, who put up with an awful lot in overseeing this work from start to finish. Special thanks go to Dr. Tom Steiman-Cameron and Dr. Bruce Smith for getting everything started and me back into astronomy (also for allowing me access to all the supercomputers). Personal thanks go to Dr. Les Tomley, who never once complained about my taking up so much of his time. But most of all I would like to thank my parents who have helped me in times and ways no one could possibly imagine.

CHAPTER 1

Introduction

Historically physicists and astronomers have striven to understand the gravitational motions of many body systems. The fundamental physics of many-body problems (called n-body problems) is well understood and has been since Newton's time. Unfortunately analytic solutions exist only for the two-body problem and the restricted three-body problem. An individual can tediously grind out solutions by hand, but if the subject of study is a galaxy (a multi-million body problem), the problem becomes overwhelming. A computer can perform the direct n-body calculations for a reasonably large n but a problem such as a galaxy is still too great even for the computer. The number of mathematical operations in the direct n-body calculation goes as n^2 and is computationally extremely inefficient. To remedy this problem many different algorithms were and still are being developed. These computer algorithms concentrate on efficient numerical methods to solve for complex aspects of the n-body calculation. An excellent overview of many such n-body algorithms is presented by J. Sellwood (1987). Once an algorithm is established, a method to test that algorithm is needed. How does one compare the computer results to analytic results for a large number of bodies or particles?

The purpose of this study is to demonstrate how an n-body computer code can be tested for use in galactic simulations.

There are many approaches to making an efficient n-body computer code. One method is to approximate the potential by discrete cells. In 2-dimensions this can be accomplished by dividing up the computational space into an L x L potential grid. The potential of the system can then be calculated very efficiently by use of the Fast Fourier Transform (FFT). From the potential, the forces on each particle are calculated. Then the velocity and position of each particle are updated numerically by using an Ordinary Differential Equation (ODE) solver. Both the FFT and ODE algorithms are very efficient and numerically well understood. These types of computer models are often referred to as FFT n-body grid codes. This thesis concentrates on testing a particular 2-D Cartesian grid FFT n-body code which is denoted as CART2D. By 2-D Cartesian we mean the potential grid consists of square cells (as opposed to a polar grid) on a two-dimensional plane. The lack of a third dimension makes CART2D ideal for studying spiral disk galaxies (elliptical or spherical galaxies would require a 3-D version). Specifics about the code CART2D are the subject of Chapter 2.

Each aspect of the FFT grid code CART2D is tested. The following chapters take an in-depth look at validating each element of the code. We first start with simple single

particle tests, then gradually increase the number of particles, finishing with over 100,000 particles in one simulation. Each test is fully described in order to serve as a reference for testing all FFT n-body codes.

The first test is a single particle potential test. This test, described in Chapter 3, is designed to validate the potential solver. A particle is placed at the center of a cell in the middle of the computational grid. The potential at each cell center is then compared with the analytic potential. The second test is a central mass two-body problem, where the orbital motion, conservation of angular momentum and energy are checked. This two-body test is described in Chapter 4. Chapter 5 presents the third test which uses 10 particles with a central core mass. Both the orbital motion and conservation laws are checked. Finally the fourth test (Chapter 6) is a Kalnajs disk (Kalnajs 1972) of 102400 particles. The Kalnajs disk is a special distribution of particle positions and velocities. Analytically the Kalnajs disk has many properties that can be examined. Numerically results can be checked against previous results in the literature (Hohl 1972, Miller 1976). Concluding remarks and overall results are discused in Chapter 7. Before we describe the tests in detail, a description of the code CART2D is presented.

CHAPTER 2

2-D Cartesian FFT Code

The 2-D Cartesian FFT Code (CART2D) was written by Tom Steiman-Cameron at NASA Ames Research Center. The code is based on the algorithm used for galactic modeling developed by Miller and Prendergast (1968). Part A explains how the potential is calculated. Part B explains how the force is calculated and how a particle's velocity and position are updated.

A) Potential Calculation

The potential for a system of discrete particles is (Danby 1988):

$$\Phi = \sum_{a<b}^{N} \sum_{b=1}^{N} G_{ab}\, m_a\, m_b \,, \qquad (2\text{-}1)$$

where:

$$G_{ab} = \frac{-k^2}{R_{ab}} = \frac{-k^2}{|R_b - R_a|} \,, \qquad (2\text{-}2)$$

and:

$R_{ab} = |R_b - R_a|$, distance between particle a and b,
m_a = mass of particle a,
m_b = mass of particle b,
N = total number of particles,
k^2 = gravitational constant.

Equation 2-1 uses the exact separation of particles a and b.

We can simplify this expression considerably if we are willing to lose some accuracy. First we choose all the particles to have equal mass:

$$m_a = m_b = m \ . \qquad (2\text{-}3)$$

In a real galaxy each star does not have the same mass. However it is not possible for us to represent each star in a galaxy by a different particle because of computer memory limitations. Therefore we think of each particle as being a cluster of stars, where each cluster has the same mass. The next approximation is to discretize the distance to each particle. This is done with a potential grid. We define a square grid of L X L unit length square cells (see Figure 2-1a). Each cell can contain any number of particles. For the potential calculation only, we assign the position of each particle in the cell to be the center of that cell. This is equivalent to having a large particle equal to the sum of the particles inside the cell, located at the center of the cell. This simplifies equations 2-1 and 2-2 considerably. If we have q particles in cell j then:

$$M_j = \sum_{i=1}^{q} m \ . \qquad (2\text{-}4)$$

The potential can now be written for an L X L grid as:

$$\Phi = \sum_{i<j}^{L} \sum_{j=1}^{L} G_{ij} M_j M_i \ , \qquad (2\text{-}5)$$

$$G_{ij} = \frac{-k^2}{s_{ij}} \ , \qquad (2\text{-}6)$$

where:

s_{ij} = distance between cell i and j (measured at cell centers).

To suppress two-body interactions and preserve numerical stability, we use a softened potential. This changes equation 2-6 to:

$$G_{ij} = \frac{-k^2}{(s_{ij}^2 + \epsilon^2)^{1/2}} \ , \qquad (2\text{-}7)$$

where:

ϵ = softening parameter.

The use of a softened potential can be the source of many problems as will be seen in section C, and throughout this thesis.

Using equation 2-7 instead of 2-6, we now employ a numerical method to quickly perform the summation over all cells in the grid for equation 2-5. This numerical method is the Fast Fourier Transform (FFT). If we take the FFT of both sides of equation 2-5, and apply the discrete Fourier convolution theorem in 2 dimensions, we get (Binney and

Tremaine 1989):

$$\hat{\Phi} = 2(L^2)\hat{G}_{ij}\hat{M}_j \quad . \tag{2-8}$$

This is the Fourier transform of the potential. To get the actual potential, we simply take the inverse FFT of the potential as calculated in equation 2-8. Equation 2-8 has many computational advantages over equation 2-1. First, G_{ij} need be calculated only once at the start of the program. The reason for this is that equation 2-7 is a function of the distances between cell centers which are fixed and known quantities. Secondly, the variable M_j is the only active component which needs to be updated each time step by equation 2-4. This is a simple and vectorizable procedure. The cost in accuracy is that we are now using a discretized softened potential.

B) Force Calculation and Orbit Integration

The force on each particle is the gradient of the potential (the potential that we are using is the inverse FFT of equation 2-8). This force is then applied to each particle. A new velocity and position are then calculated by an ODE solver. To calculate the force on each particle, we approximate the force by a polynomial expansion for the derivative of the potential. The coefficients of this expan-

sion can be expressed as making a least-squares fit to the gradient of the potential (Hockney and Eastwood 1988). This is accomplished by fitting the gradient with the potentials of a cell's nearest neighbors (see Figure 2-1b). The resulting functions for the force in both X and Y are:

$$\begin{aligned} F_x = {}^1/_{12}[&(-2 + 4\rho_x + 3\rho_y)P_1 - 8\rho_x P_2 + (2 + 4\rho_x - 3\rho_y)P_3 \\ &+(-2 + 4\rho_x)P_4 - 8\rho_x P_5 + (2 + 4\rho_x)P_6 + (-2 + 4\rho_x - 3\rho_y)P_7 \\ &- 8\rho_x P_8 + (2 + 4\rho_x + 3\rho_y)P_9] , \end{aligned} \tag{2-9}$$

$$\begin{aligned} F_y = {}^1/_{12}[&(-2 + 3\rho_x + 4\rho_y)P_1 + (-2 + 4\rho_y)P_2 \\ &+ (-2 - 3\rho_x + 4\rho_y)P_3 - 8\rho_y P_4 - 8\rho_y P_5 - 8\rho_y P_6 \\ &+ (2 - 3\rho_x + 4\rho_y)P_7 + (2 + 4\rho_y)P_8 \\ &+ (2 + 3\rho_x + 4\rho_y)P_9] , \end{aligned} \tag{2-10}$$

where:

ρ_x = x-component of particle's distance from cell center
ρ_y = y-component of particle's distance from cell center
P_i = Potential in cell i

The force is calculated for each individual particle and the particle's velocity and position are then updated. This code uses a Leap-Frog, or Euler-Cromer Half-Step Approximation (HSA) ODE solver (Cromer 1981). Velocities are calculated on the half time step, while the positions are updated

on the time step. Therefore in vector notation, a particle's velocity and position are calculated as:

$$\mathbf{V}_i^{(n+1/2)} = \mathbf{V}_i^{(n-1/2)} + \mathbf{F}_i^{(n)} \Delta t \quad , \tag{2-11}$$

$$\mathbf{R}_i^{(n+1)} = \mathbf{R}_i^{(n)} + \mathbf{V}_i^{(n+1/2)} \Delta t \quad , \tag{2-12}$$

where:

n = current time step,
$\mathbf{V}_i^{(n-1/2)}$ = velocity (in X and Y) of particle i at the previous half time step n-1/2,
$\mathbf{R}_i^{(n)}$ = position (in X and Y) of particle i at the current time step n,
$\mathbf{F}_i^{(n)}$ = force (in X and Y) on particle i at the current time step n,
Δt = 1, time step increment.

Even though the potential is calculated as if all the particles were located at the center of a cell, equations 2-11 and 2-12 show that the force is calculated for the actual position of the particle.

Cromer (1981) states that the Half-Step Algorithm is a moderately stable ODE solver. However if the forces become too large the algorithm breaks down and is not accurate. This is just one reason we use a softened potential. If two particles are very close together they will have a large force and consequently a large velocity may result. Softening the potential in equation 2-7 fixes this problem. The second reason we use a softened potential is that according

to Mihalas (1968), for galaxies, a two-body collision is extremely rare to non-existent. Therefore to have a collisionless system, we soften the potential.

C) Image Force: The Effects of a Softened Potential

One of the disadvantages of a softened potential is the creation of a non-physical numerical force. This is strictly an artifact of the numerical scheme. Since the potential is calculated as if all the particles in one cell were at the center of that cell, the force that is applied to the particle's actual position is then biased towards the center of that cell. This is best illustrated by examining a single particle in one cell. Each square cell has unit length in the x-direction and unit length in the y-direction. For this example we define the center of the cell to be the origin, and the limits of the cell are at [-0.5, +0.5] for x and y. We place the particle on the x-axis a distance of +0.4 from the origin. The initial velocity of this particle is zero (particle at rest). The results of this example are shown in Figure 2-2. Figure 2-2 shows by the first time step the particle has already moved through the origin along the x-axis. After three time steps the particle has reversed direction and is approaching the origin. By six time steps the particle has completed one

period. Although there are no external forces on the particle due to other particles or imposed potentials, the particle acts as a harmonic oscillator. Since the potential is calculated as if the particle were at the center of the cell the particle sees a mirror image of itself at the center of the cell. This image particle attracts the test particle and starts the harmonic motion. Figure 2-3 is a vector force plot showing the magnitude and direction of this image force. Notice that all the vectors point to the center of the cell.

CHAPTER 3

Single Particle Potential Test

A simple test of the potential solver is to place a single particle at the center of the computational grid, then compare the potential of the code to the analytic potential. To avoid ambiguities due to the grid, the values of the potential comparison are performed at the cell centers. The analytic potential must include the softening parameter. For a single particle at the middle of the grid, the potential at a cell center is:

$$\Phi(i)_{analytic} = \frac{W_1}{(\epsilon^2 + r_i^2)^{1/2}} . \qquad (3\text{-}1)$$

Equation 3-1 is the analytic softened potential for cell i. W_1 is a constant (explained below in Chapter 4), epsilon is the softening parameter, and r_i is the radial distance from the particle to the center of cell i.

Figure 3-1 shows the percent difference between the computed potential and the analytic potential for a single particle. For this case the computational grid is 16 X 16, W_1 is set to unity, and epsilon squared is 0.25. The dashed contour is a difference of 1.0×10^{-5} %, while the solid contour is a 2.0×10^{-5} % difference. However one cannot help

but notice the four-fold symmetry of Figure 3-1. It is possible that the symmetry is due to the Cartesian nature of the grid, and the 2D FFT algorithm. Figure 3-2 uses the same initial conditions, except the grid size has been increased to 64 X 64. The contour levels are the same as in Figure 3-1. Again the errors are on the order of 2.0×10^{-5} percent. This test assures us that the potential solver functions to 2×10^{-5} percent of the analytic values for a single particle.

CHAPTER 4

Two-Body Central Mass Test

This validation exercise consists of a single particle in a central force. All the tests in this section consider a particle in a circular orbit about a massive core. For this test, we have incorporated in CART2D a central potential to mimic a massive core at the center of the grid. In contrast to the single particle potential test where only the potential solver was utilized, the two-body test uses the combination of the potential solver and the particle pusher. This section not only validates the ODE solver, but the integration of the potential solver with the ODE particle pusher for a single particle. In Part A) we show how to set up a run by calculating the variable for the gravitational constant called W_1. Part B) computes the orbital path and compares the results with that of a true circle. Part C) examines the conservation of angular momentum, and Part D) shows how well energy is conserved. The last section (Part E) summarizes the results of the two-body test.

A) Calculation of W_1

The constant W_1 represents the gravitational constant for the run. This variable is used to scale the simulation in time, mass, and distance. If it is improperly calculated

the results of a computer run will be useless and non-physical. The calculation is relatively simple, but is different for each type of spatial load or problem. Although the value of W_1 is necessary to make a run, it does not describe any physics of the simulation, and its calculation is usually ignored in journal publications. That is why we emphasize its importance and present the calculation here.

a) Definition of W_1

To show how time, mass, and distance affect W_1, we present its definition. To accomplish this we make the distinction between physical units and program units. Therefore we define the following:

small letters	= physical units
CAPITAL letters	= program units
BOLD Capital letters	= conversion between physical and program units

With this in mind we define:

mass	= m	= **M**	M
distance	= r	= **R**	R
time	= t	= **T**	T
velocity	= v	= **V**	V
force	= f	= **F**	F

Force is defined by:

$$f = \frac{k\, m\, m_c}{r^2} = ma, \qquad (4\text{-}1)$$

where:
- k = Newtonian Gravitational constant
- m = mass of a particle
- m_c = mass of the central core
- a = acceleration
- r = radial distance of particle from central core mass.

Converting equation 4-1 into program units:

$$f = FF = \frac{kMMMM_c}{R^2R^2}, \tag{4-2a}$$

$$ma = MM\frac{d^2r}{dt^2} = MM\frac{d^2(RR)}{[d(TT)]^2}. \tag{4-2b}$$

Therefore:

$$\frac{kM^2}{R^2}\frac{MM_c}{R^2} = \frac{MR}{T^2}M\frac{d^2R}{dT^2}. \tag{4-3}$$

So:

$$\frac{kMT^2}{R^3}\frac{MM_c}{R^2} = M\frac{d^2R}{dT^2}. \tag{4-4}$$

We now define the program constant W_1 to be:

$$W_1 = \frac{kMT^2}{R^3}. \tag{4-5}$$

W_1 is the conversion of the gravitational constant into program units. It literally scales the computer simulation. The final equation of motion is:

$$\frac{W_1 M_c}{R^2} = \frac{d^2R}{dT^2}. \tag{4-6}$$

b) Application to the two-body problem

For a particle in a circular orbit around a core mass with constant circular velocity v_c the force is:

$$f = ma = \frac{mv_c^2}{r}. \tag{4-7}$$

Equating with equation 4-1, we obtain

$$v_c^2 = \frac{km_c}{r} . \qquad (4\text{-}8)$$

Converting to program units yields,

$$v_c^2 = V^2 V_c^2 = \frac{R^2}{T^2} V_c^2 = \frac{kMM_c}{RR} . \qquad (4\text{-}9)$$

Therefore

$$V_c^2 = \frac{kMT^2}{R^3} \frac{M_c}{R} . \qquad (4\text{-}10)$$

Substituting in the definition of W_1 from equation 4-5:

$$V_c^2 = \frac{W_1 M_c}{R} . \qquad (4\text{-}11)$$

This can easily be solved for W_1 if the circular velocity is known. If not, we need a second equation for the circular velocity. In program units we define the circular velocity to be the circumference divided by the number of time steps for the particle to complete one revolution around the core mass (period). Therefore in program units, circular velocity is:

$$V_c = \frac{2\pi R}{T} , \qquad (4\text{-}12)$$

where T is the period at radius R in program units (steps). Squaring the expression in equation 4-12 and equating it with the right hand side (RHS) of equation 4-11 we get:

$$\frac{4\pi^2 R^2}{T^2} = W_1 \frac{M_c}{R} . \qquad (4\text{-}13)$$

Solving for W_1:

$$W_1 = \frac{4\pi^2 R^3}{T^2 M_c} . \qquad (4\text{-}14)$$

From equations 4-11 and 4-14 we can calculate the value of W_1. Equation 4-14 is used when we want to define a period for the simulation. Although sometimes unavoidable, using the period to calculate W_1 is not always the preferred option. When running the simulation we must keep in mind that the particle position is updated by an Euler-Cromer HSA (see Chapter 2). It is therefore desirable for numerical stability to keep the particle velocity under a grid-cell per time step. As a rule of thumb we use:

$$V_c = 0.75 \ (\text{grid cells} / \text{time step}) . \qquad (4\text{-}15)$$

From equation 4-12 we may find T if necessary.

B) Orbital Trajectory Validation

a) Initial Conditions

The object of this test is to see how well the trajectory of a particle placed in a circular orbit about a core mass compares to that of an actual circle. For this simulation, unless otherwise stated, we use a grid of 256 X 256.

A core mass potential equivalent to 1000 particles is located at the middle of the grid (X = 128.5, Y = 128.5). The initial position of the test particle is along the x-axis directly to the right of the core mass at a radial distance of 64 (X = 192.5, Y = 128.5). If we define the period to be 540 time steps we can use equation 4-14 to get the value for $W_1 = 3.549X10^{-2}$. The initial circular velocity is then calculated using equation 4-11. Since the ODE solver uses a half-step Euler-Cromer algorithm the initial velocity must be assigned at the previous half-step, that is at t = -1/2. In Cartesian coordinates the initial half-step circular velocity is: $V_x = 4.33x10^{-3}$, $V_y = 0.74466$. These initial conditions are inputs for seven runs in which the softening parameter squared (from here on called ASQ) has the values 2.00, 1.00, 0.75, 0.50, 0.25, 30.00 and a run in which the mass of the orbiting particle is removed.

b) Results

First we performed a test in which the mass of the orbiting particle was removed from the potential calculation. This way we remove any effects caused by the image force. Since there is a core potential at the center of the grid, this specific run is a good test for the ODE integrater. Figure 4-1 shows the particle's orbital path about the core potential. Figure 4-2 gives the percent error of the

orbital path against a true circle of radius 64. Figure 4-2 shows a maximum error of 0.04 percent. This means that the particle's orbit is at most 1/40th of a grid box from a true circle. This is judged an acceptable error, especially when we examine the results where the particle's mass is included.

Including the particle's mass in the potential calculation enables the image force to have a dramatic influence on the results. We must now include the softening parameter as a variable in our runs. Figures 4-3 through 4-7 present the orbital paths for the initial conditions as stated in Section a. The mass of the particle is now included and the value for ASQ takes the values 2.00, 1.00, 0.75, 0.50, and finally 0.25. For values of ASQ between 2.00 and 0.50 (Figures 4-3 through 4-6), the trajectory is around the core mass, but clearly is not circular. Figure 4-8 shows, for these values of softening, the percent difference from a true circle. Examination of this graph shows errors as large as 35 percent for ASQ = 0.50, and 25 percent for ASQ = 2.00. In Figure 4-7 we use an ASQ = 0.25 and the resulting trajectory is linear. The cause of all these discrepancies is not an error in the code, but is due to the image force as discused earlier in Chapter 2. This is explained in detail in the following section.

c) Effects of softening

As mentioned earlier a softened potential can cause problems; however, the magnitude of these problems on CART2D is not known. Recalling the force plot of Figure 2-3 where no core mass is present, we can see that the force points to the center of the grid and grows in magnitude as the particle approaches the edge of the cell. Figures 4-9 through 4-13 are force plots of the initial grid cell for the runs shown in Figures 4-3 through 4-7. Examination of these force plots shows that as ASQ decreases, the force vectors point more and more towards the center of the cell. Normally we would expect all the force vectors for Figures 4-9 to 4-13 to point to the core mass (to the left) and be close to parallel. The reason the force vectors are not parallel is that the magnitude of the image force produced increases as ASQ decreases. Eventually the magnitude of the image force exceeds the magnitude of the central force (from the core mass) and the net force points towards the center of the grid box (that is why the focus of the force vectors is along the x-axis).

To prove this hypothesis we can theoretically place a single particle in a grid cell subject to a central core potential at a distance of r. Figure 2-1b shows the placement of the particle in a grid cell. If we define:

r = radius from core mass M_c (x, y)
ρ = radial distance of particle from cell center(ρ_x,ρ_y)
$\Phi_c(i)$ = potential from core mass M_c in cell i
$\Phi_p(i)$ = potential from the particle in cell i
s = distance between cell centers (s_x, s_y)
ϵ = softening parameter
P_i = total potential in cell i

The potential from the core mass is then:

$$\Phi_c(i) = \frac{k M_c}{r_i} = \frac{k M_c}{(x_i^2 + y_i^2)^{1/2}} , \qquad (4\text{-}16)$$

and the potential from the particle is:

$$\Phi_p(i) = \frac{k}{(s^2 + \epsilon^2)^{1/2}} = \frac{k}{(s_x^2 + s_y^2 + \epsilon^2)^{1/2}} . \qquad (4\text{-}17)$$

Therefore, the total potential in cell i is:

$$P_i = \Phi_c(i) + \Phi_p(i) . \qquad (4\text{-}18)$$

Using the force relations described in equations 2-9 and 2-10, where the potentials (P_i) can be represented as equation 4-18, we have (see Figure 2-1b):

$$P_1 = \frac{k M_c}{[(x-1)^2 + (y-1)^2]^{1/2}} + \frac{k}{[(-1)^2 + (-1)^2 + \epsilon^2]^{1/2}} , \qquad (4\text{-}19)$$

$$P_2 = \frac{k M_c}{[x^2 + (y-1)^2]^{1/2}} + \frac{k}{[(-1)^2 + \epsilon^2]^{1/2}} , \qquad (4\text{-}20)$$

$$P_3 = \frac{k M_c}{[(x+1)^2 + (y-1)^2]^{1/2}} + \frac{k}{[(1)^2 + (-1)^2 + \epsilon^2]^{1/2}}, \qquad (4\text{-}21)$$

$$P_4 = \frac{k M_c}{[(x-1)^2 + y^2]^{1/2}} + \frac{k}{[(-1)^2 + \epsilon^2]^{1/2}}, \qquad (4\text{-}22)$$

$$P_5 = \frac{k M_c}{[x^2 + y^2]^{1/2}} + \frac{k}{\epsilon}, \qquad (4\text{-}23)$$

$$P_6 = \frac{k M_c}{[(x+1)^2 + y^2]^{1/2}} + \frac{k}{[(1)^2 + \epsilon^2]^{1/2}}, \qquad (4\text{-}24)$$

$$P_7 = \frac{k M_c}{[(x-1)^2 + (y+1)^2]^{1/2}} + \frac{k}{[(-1)^2 + (1)^2 + \epsilon^2]^{1/2}}, \qquad (4\text{-}25)$$

$$P_8 = \frac{k M_c}{[x^2 + (y+1)^2]^{1/2}} + \frac{k}{[(1)^2 + \epsilon^2]^{1/2}}, \qquad (4\text{-}26)$$

$$P_9 = \frac{k M_c}{[(x+1)^2 + (y+1)^2]^{1/2}} + \frac{k}{[(1)^2 + (1)^2 + \epsilon^2]^{1/2}}, \qquad (4\text{-}27)$$

If we now fix the position of the particle at (ρ_x, ρ_y) from the center of the cell, we can find the component of the force (in x, and y) due to the softening parameter only. Examining the $\Phi_p(i)$ component of P_i we can see that:

$$\Phi_p(1) = \Phi_p(3) = \Phi_p(7) = \Phi_p(9) = \frac{k}{(\epsilon^2 + 2)^{1/2}} = \omega_1 \ , \qquad (4\text{-}28)$$

and also that:

$$\Phi_p(2) = \Phi_p(4) = \Phi_p(6) = \Phi_p(8) = \frac{k}{(\epsilon^2 + 1)^{1/2}} = \omega_2 \ . \qquad (4\text{-}29)$$

For consistency we define

$$\Phi_p(5) = \frac{k}{\epsilon} = \omega_5 \ . \qquad (4\text{-}30)$$

Substituting all these equations into equation 2-9 and 2-10 yield:

$$\begin{aligned} F_x = {}^1/_{12}[&(-2+4\rho_x+3\rho_y)\Phi_c(1) - 8\rho_x\Phi_c(2) + (2+4\rho_x-3\rho_y)\Phi_c(3) + \\ &(-2+4\rho_x)\Phi_c(4) - 8\rho_x\Phi_c(5) + (2+4\rho_x)\Phi_c(6) + \\ &(-2+4\rho_x-3\rho_y)\Phi_c(7) - 8\rho_x\Phi_c(8) + (2+4\rho_x+3\rho_y)\Phi_c(9) + \\ &(-2+4\rho_x+3\rho_y)\omega_1 - 8\rho_x\omega_2 + (2+4\rho_x-3\rho_y)\omega_1 + \\ &(-2+4\rho_x)\omega_2 - 8\rho_x\omega_5 + (2+4\rho_x)\omega_2 + \\ &(-2+4\rho_x-3\rho_y)\omega_1 - 8\rho_x\omega_2 + (2+4\rho_x+3\rho_y)\omega_1] \ , \end{aligned} \qquad (4\text{-}31)$$

and

$$F_y = {}^1/_{12}[(-2+3\rho_x+4\rho_y)\Phi_c(1) + (-2+4\rho_y)\Phi_c(2) + \quad (4\text{-}32)$$
$$(-2-3\rho_x+4\rho_y)\Phi_c(3) - 8\rho_y\Phi_c(4) - 8\rho_y\Phi_c(5) + - 8\rho_y\Phi_c(6) +$$
$$(2-3\rho_x+4\rho_y)\Phi_c(7) + (2+4\rho_y)\Phi_c(8) + (2+3\rho_x+4\rho_y)\Phi_c(9) +$$
$$(-2+3\rho_x+4\rho_y)\omega_1 + (-2+4\rho_y)\omega_2 + (-2-3\rho_x+4\rho_y)\omega_1 -$$
$$8\rho_y\omega_2 - 8\rho_y\omega_5 - 8\rho_y\omega_2 + (2-3\rho_x+4\rho_y)\omega_1 +$$
$$(2+4\rho_y)\omega_2 + (2+3\rho_x+4\rho_y)\omega_1] \ .$$

These equations can be simplified to the following expressions:

$$F_x = {}^1/_{12}[\,Q_x + 8\rho_x q_\epsilon\,] \ , \quad (4\text{-}33)$$

and

$$F_y = {}^1/_{12}[\,Q_y + 8\rho_y q_\epsilon\,] \ . \quad (4\text{-}34)$$

where:

$$Q_x = (-2+4\rho_x+3\rho_y)\Phi_c(1) - 8\rho_x\Phi_c(2) + (2+4\rho_x-3\rho_y)\Phi_c(3) + \quad (4\text{-}35)$$
$$(-2+4\rho_x)\Phi_c(4) - 8\rho_x\Phi_c(5) + (2+4\rho_x)\Phi_c(6) +$$
$$(-2+4\rho_x-3\rho_y)\Phi_c(7) - 8\rho_x\Phi_c(8) + (2+4\rho_x+3\rho_y)\Phi_c(9) \ ,$$

$$Q_y = (-2+3\rho_x+4\rho_y)\Phi_c(1) + (-2+4\rho_y)\Phi_c(2) + \quad (4\text{-}36)$$
$$(-2-3\rho_x+4\rho_y)\Phi_c(3) - 8\rho_y\Phi_c(4) - 8\rho_y\Phi_c(5) -$$
$$8\rho_y\Phi_c(6) + (2-3\rho_x+4\rho_y)\Phi_c(7) + (2+4\rho_y)\Phi_c(8) +$$
$$(2+3\rho_x+4\rho_y)\Phi_c(9) \quad .$$

The variables Q_x and Q_y are the force components (x and y, respectively) due to the core mass potential and are a function of the particle's position within the cell. The variable q_ϵ is the component due to the softened potential, that is:

$$q_\epsilon = 2\omega_1 - \omega_2 - \omega_5 \quad . \quad (4\text{-}37)$$

Substituting for ω_1, ω_2, and ω_5 from equations 4-28, 4-29, and 4-30, we get:

$$q_\epsilon = \frac{2k}{(\epsilon^2+2)^{1/2}} - \frac{k}{(\epsilon^2+1)^{1/2}} - \frac{k}{\epsilon} \quad . \quad (4\text{-}38)$$

Equation 4-38 describes the multiplier (q_ϵ) of the force due strictly to the softening parameter (force is $8\rho q_\epsilon$). Figure 4-14 is a graph of q_ϵ as a function of the softening parameter. Examination of this graph indeed shows that as epsilon decreases, the force component due to softening increases, thus proving our hypothesis. The negative values show that

this is an attractive force.

Figure 4-14 is very important. It shows that if we are not careful in selection of the softening parameter the image force dominates the simulation and can lead to erroneous results. Figure 4-14 also suggests that we can minimize the image force if we use a large epsilon. This is in slight contradiction to Figure 4-8 which shows for ϵ^2 of 2.00 the percent errors are greater than that for an ϵ^2 of 0.75. However, we must be aware that as the force vector graphs (figures 4-9 through 4-13) demonstrate, the magnitude of the image force increases as the particle gets closer to the cell edge. This is also shown by the ρ multiplier to q_ϵ in equations 4-33 and 4-34 where ρ_x and ρ_y are the offset from center. As the particle enters a new cell at each time step, the effect of the image force is proportional to the distance of the particle to the center of that cell. Therefore it is possible for fortuitous choices of initial conditions and for a limited number of time-steps to have small errors for small values of epsilon. This explains the apparent inconsistencies of Figure 4-8.

Figure 4-14 indicates if we set epsilon to a very large number, the effect of the image force is minimized and we should get small errors in orbital path. Figure 4-15 shows the orbital trajectory of our particle around the core mass with an ASQ equal to 30. Figure 4-16 shows the percent

error of this path which is comparable to that of Figure 4-2, in which the particles mass was removed and hence no image force exists. Figure 4-16 shows that we get 0.2 percent error which corresponds to about 1/8th of a grid cell from that of a true circle. This is not as good as the results shown in Figure 4-2, but much better than any of the other runs in which softening is included (see Figure 4-8). Figure 4-17 shows the force vectors inside the starting cell; the vectors are essentially parallel and of equal magnitude as expected if no image force problems exist.

This result might imply that all simulations should use large values for ASQ. This is, however, a dangerous conclusion. A largely softened potential is dramatically different from a Newtonian 1/r potential.

C) Conservation of Angular Momentum

a) Initial Conditions

This section presents results from the runs made in section B, so the initial conditions for this section are identical to those in section B. This enables the reader to examine the orbital trajectories of the particle for the angular momenta discused in this section.

b) Results

The object of this test is to see how well the code

CART2D conserves angular momentum for a particle in orbit about a core mass. To show this, we plot the orbiting particle's angular momentum versus time. If angular momentum is being conserved there should be no change in angular momentum as a function of time. Since the velocities and positions of the particles are calculated by a HSA Euler-Cromer Algorithm, the angular momentum is calculated at the half time step as:

$$h^{(n+1/2)} = \frac{r^{(n)} \times r^{(n+1)}}{\Delta t} , \qquad (4\text{-}39)$$

where:

n = current time step,
$r^{(n)}$ = position at current time step,
$\Delta t = 1$, time step increment.

In two-dimensions this reduces to:

$$h = x^{(n)} y^{(n+1)} - y^{(n)} x^{(n+1)} , \qquad (4\text{-}40)$$

where x, and y are the Cartesian components of the distance from the particle to the core mass. Since the core mass is fixed on the grid, it has zero angular momentum.

Figure 4-18 shows the angular momentum for the case in which the particle's mass is removed from the potential calculation (as described in section B). The average value

for angular momentum for this case is 47.656 with a standard deviation of 1.27×10^{-3} .

Figure 4-19 plots the angular momentum of the orbiting particle for the values of ASQ = 2.00, 1.00, 0.75, and 0.50. As might be expected, softening dramatically affects angular momentum. This is due to the image force exerting a torque on the particle. Table 4-1 contains the average values and standard deviations of angular momentum for all the runs. For an ASQ = 0.25 (linear trajectory: Figure 4-7) the standard deviation is 28.8 and Figure 4-20 shows the angular momentum actually changes sign. Considering the orbital trajectories presented in part B, this should come as no surprise. Likewise if we use an ASQ = 30, we would expect excellent results similar to Figure 4-18. Figure 4-21 shows the angular momentum for a value of ASQ = 30, and indeed the results are excellent with an average value of 47.63, and a standard deviation of 7.604×10^{-3} which is comparable to Figure 4-18 where the particle has no mass.

D) Conservation of Energy

a) Initial Conditions

The results in this section are obtained from the runs made in Part B. Therefore, this section uses the same initial conditions as Part B.

b) Results

The object of this test is to see how well the code CART2D conserves energy for a particle in orbit about a massive core. As in Part B, we plot the orbiting particle's energy vs. time. If energy is being conserved there should be no change as a function of time. The total energy is approximated as the sum of the potential at the particle's grid center and an average of the kinetic energies at each integer time step. That is:

$$E = P_5 + [(v_x^{(n)})^2 + (v_y^{(n)})^2 + (v_x^{(n+1)})^2 + (v_y^{(n+1)})^2]/4 \ , \qquad (4\text{-}41)$$

where v_x and v_y are the x and y components of the velocity and P_5 is the potential energy in the center of the cell.

For the run in which the mass of the particle was removed in order to eliminate the effects of the image force, we get an average energy of -0.27719, and a standard deviation of 2.44×10^{-3}. Figure 4-22 plots the energy for this particular run. When softening is included we get results as shown in Figures 4-23 and 4-24. Table 4-2 presents the average and standard deviations of the energy corresponding to Figures 4-22 through 4-25. Figure 4-25 is the plot of total energy for ASQ = 30. As in previous sections, the results for ASQ = 30 are comparable to that of not including

the particle's mass in the potential calculation.

E) Summary

The purpose of this chapter was to describe a simple test where the results can be compared to the expected values. In Chapter 3 we found the potential was accurate to 2×10^{-5} percent. Here we find the ODE integrater working to within 0.04 percent or 1/40th of a grid cell (this is when the particle's mass was removed from the potential calculation). When we include the mass of the orbiting particle in the potential calculation we find even larger errors resulting from the image force.

The two-body problem for FFT grid codes is most demanding. This chapter points out the shortcomings of FFT grid codes by demonstrating the effect of the image force on an orbiting test particle. To minimize the effect of this image force, we use a very large value for the softening parameter. Once this is done, the code passes the two body problem test (see all the figures and tables for ASQ = 30). Using a large value for the softening parameter works well for the two-body problem because there are no other orbiting particles. However, using very large values for the softening parameter can yield inaccurate results for simulations with many particles.

CHAPTER 5

The Ten-Body Problem Test

This test is an extension of the previous two-body problem test. Instead of a single particle in a circular orbit about a massive core, this test uses nine particles in near circular orbits about a massive core. The existence of additional particles means the existence of additional forces which will cause deviations in orbital trajectory. Therefore circular orbits are not expected and can not be used for test purposes. The purpose of this exercise is to determine if angular momentum and energy are being conserved. As in Chapter 4, we use a positionally fixed core potential to mimic a massive particle at the center, and as in the previous chapter we use many of the same initial conditions (grid size and core mass). Part A covers the calculation of W_1 and part B reviews the initial conditions of each run. Finally part C presents the results of both the conservation of angular momentum and energy.

A) Calculation of W_1

The spatial and velocity load for the ten-body test is very similar to that of the two-body problem. Since we are looking at near circular orbits the calculations for W_1 are the same as in Chapter 4. We want the velocity of the fast-

est particle to be that of equation 4-15. The fastest particle is the innermost particle (R_{in}), that is the particle closest to the core mass. Therefore using equation 4-15 we have:

$$V_c(R_{in}) = 0.75 , \qquad (5\text{-}1)$$

and using equation 4-11, we can calculate W_1 as:

$$W_1 = \frac{R_{in} V_c^2(R_{in})}{M_c} . \qquad (5\text{-}2)$$

The value used for R_{in} and all the other radii are discussed in part B, the initial conditions.

B) Initial Conditions

The initial conditions for the ten-body test are as follows. The grid size is the same as in the two-body test which is a 256 X 256 grid. The core mass is that of 1000 particles located at X = 128.5, and Y = 128.5 (same as in chapter 4). Nine particles are placed about the core mass, each at a radius 10 units farther than the previous particle. Each particle is then displaced 90 degrees counter clockwise from the previous particle. The radius of the innermost particle, R_{in} is 10 (X = 138.5 and Y = 128.5). The radius of the next particle is 20 and it is located 90 degrees counter clockwise. So its initial position is X=

128.5, Y = 148.5. The initial velocities are calculated using equation 4-11 where W_1 is defined for R_{in}, and R is just the radial distance of the particle from the core mass. Since circular orbits are not expected, we initially place the particles in slightly elliptical orbits. This is done by using equation 4-11 to calculate the velocity of each particle, but instead of applying that velocity to the previous half-step, we simply apply that velocity to the inital position. This results inititally in near circular orbits. To help clarify the initial load, Table 5-1 shows the initial positions and velocities of all nine particles.

C) Results

The results of runs made with the initial conditions described above are presented in Figures 5-1 through 5-5. As in Chapter 4, the softening parameter (ϵ) is varied and the results do show a dependence on softening. Figure 5-1 is an orbital trajectory plot for a value of ASQ = 5.00 (ASQ = ϵ^2). We observe that the orbital paths of all the particles are nearly circular. Perfect circular motion is not expected because of the additional forces from the other particles. Comparing to Figure 5-2 where ASQ = 0.25, we see the effect of the image force on particle orbits. As in Chapter 4 (Figure 4-7), three of the outermost particles have linear orbits, caused by the image force due to a small

value of softening.

a) Angular Momentum

Figure 5-3 and 5-4 show the total angular momentum for the entire system. Both of these plots show that as ASQ becomes less than or equal to one, total angular momentum is not conserved. Table 5-2 presents the statistics for the conservation of angular momentum test. Comparing standard deviations (for ASQ greater than one) with the two body test shows better conservation of angular momentum for the ten body test. This is because the particles in the ten body test are generally closer to the core mass where the image force does not have as large an effect (compared to the outer regions).

b) Conservation of Energy

Figure 5-5 shows the total energy of the ten body system as a function of time. Examination of Figure 5-5 shows that as time changes the total energy oscillates with the period of the innermost particle. This is to be expected because the inner particles dominate the energy. Since there is no secular change in energy, we can say that energy is being conserved for $\epsilon^2 > 2$.

CHAPTER 6

The Kalnajs Disk Test

To this point, we have outlined and demonstrated tests that use only a small number of particles. Although these tests are satisfactory for verifying certain aspects of the code, they do not test the code for what it is designed for, a galactic disk. For this test we have selected a Kalnajs disk. In this chapter we consider a 102400 particle disk without any core mass or any additional potentials (i.e., a halo or perturbing potential). The Kalnajs disk and the omega models, described by A. J. Kalnajs (1972), have been the subject of many numerical experiments. The results of these computer experiments stand as verification benchmarks, specifically those of Hohl (1972) and Miller (1976). Since exact orbits of particles can not act as verification measures, we study global aspects of the simulation. For example, in this chapter we will look at the global structure of the Kalnajs disk as a function of time, as well as the stability of the disk, its surface density, and its velocity dispersion. We will show how each disk is set up and loaded. Also the techniques for measuring specific quantities for the disk are discussed.

The purpose of this chapter is to compare previously obtained numerical results for a system having a large num-

ber of particles with the results from CART2D. However, one is to be warned in making comparisons between numerical results. Sellwood (1983) states for Kalnajs disks he has studied, results can differ on the order of 50% in stability when different techniques are used to load the same disks. With this in mind, we look to see if global characteristics of the Kalnajs disk are reproducible.

A) The Kalnajs Omega Models

A. J. Kalnajs (1972) described a set of self-gravitating theoretical disks which have different stability criteria. The surface density of a Kalnajs disk is:

$$\Sigma(r) = \Sigma_o(1 - r^2/R_{max}^2)^{1/2} \ . \qquad (6\text{-}1)$$

If initially given a constant angular rotation rate Ω_0, the Kalnajs disk will act as a rigid rotator, i.e., the inner regions rotate at the same angular velocity as the outer regions (a spoked wheel is a good example). The angular velocity is defined to be:

$$\Omega_0 = (\pi^2 G \Sigma_o / 2 R_{max})^{1/2} \ . \qquad (6\text{-}2)$$

This special case of a rigid rotating disk has no velocity dispersion and is called a cold disk. If we increase the

velocity dispersion, or heat the disk up, the mean angular velocity Ω slows down. If we constrain $0 \leq \Omega \leq \Omega_0$ we can have a set of disks each with different mean rotation rates, and velocity dispersions. This set of Kalnajs disks are often referred to as the Kalnajs Disk Omega Models. The velocity dispersion is calculated as (Hohl 1972):

$$\sigma_\phi = \sigma_r = [\,(\Omega_0^2 - \Omega^2)\,(R_{max}^2 - r^2)/3\,]^{1/2}\ , \qquad (6\text{-}3)$$

for Gaussian statistics. Equation 6-3 states one of many interesting properties of the Kalnajs disk, that is the azimuthal velocity dispersion is equal to the radial velocity dispersion.

Now that we have described what the omega models are, we can show how they can be used to test an N-body code for galactic simulations. Each omega model is characterized by its stability to perturbations. The cold disk or Ω_0 model is very unstable. The slightest perturbation, such as the grid approximation to the potential, will cause the disk to break up. In fact, all the omega models are unstable. We can show regions of instability by examining the modes of each omega model. To show what a mode is, we use a function f that is dependent on position and velocity (phase space) such that (in Cartesian space):

$$dm = f(x,y,z,v_x,v_y,v_z)\,dv_x\,dv_y\,dv_z\,dx\,dy\,dz\ , \qquad (6\text{-}4)$$

where dm corresponds to the mass in the phase space region $dv_x dv_y dv_z dx dy dz$. We call $f(x,y,x,v_x,v_y,v_z)$ the time independent distribution function. The distribution function used with the collisionless Boltzmann equation:

$$\frac{\partial f}{\partial t} + \sum_{i=1}^{3} \left(v_i \frac{\partial f}{\partial x_i} - \frac{\partial \Phi}{\partial x_i} \frac{\partial f}{\partial v_i} \right) = 0 \ , \qquad (6\text{-}5)$$

can tell us much about the stability of a specific distribution function. Solutions to equation 6-5 are sets of orthogonal eigenvectors, or normal modes. Each mode can be stable or unstable to perturbations, where the lower order modes (m=1, 2, 3, ...) can dramatically affect the stability of the entire system in a very short time. The higher order modes generally do not have as dramatic an effect in the same amount of time.

For the Kalnajs disk, the modes are functions of the associated Legendre polynomials and therefore the modes have an (n,m) dependence, were m is the azimuthal mode. The azimuthal modes are of interest because they show up as lobes in the density of the disk (the n values do not). For example, an m = 2 mode will show up as two dense regions of particles located about the center of the disk. An m = 3 mode will have three dense regions located about the center of the disk, an m = 4 will have four regions and so on. Figure 6-1 is taken from Kalnajs (1972), and shows where

each mode is unstable as a function of the omega used (solid lines are regions of instability). As can be seen from this chart, all the omega models are unstable in some region. However for $\Omega \leq 0.4\Omega_0$ all of the lower order modes are stable (the (3,1) mode extends slightly beyond $0.4\Omega_0$). It is not until one reaches the 3rd set of modes that instability is present.

The result of one such instability, the m=2 or bar mode instability, shows up in a disk as a barred spiral type structure. Figure 6-1 shows that omega models that are less than or equal to $0.4\Omega_0$ should be stable to this bar mode. This is the basis for our test.

B) Published Numerical Experiments

Hohl (1972) published results for a set of Kalnajs disk omega models. He used a Cartesian grid code similar to CART2D with a 128 X 128 potential grid. Figure 6-2 is taken from Hohl (1972) and presents his published results for $\Omega / \Omega_0 = 0.8$, 0.6, 0.4, and 0.0. Figure 6-2 is subdivided into 4 separate figures each containing specific results for the omega models Hohl tested. Figure 6-2a presents the particle plots of the omega models, Figure 6-2b presents the corresponding Q values, 6-2c the radial velocity dispersion, and 6-2d the surface density, all as functions of time. The times indicated in Hohl's results are the periods, or the

number of revolutions the disk has made. One period is the amount of time it takes the cold disk or Ω_0 case to complete one revolution. Hohl indeed found that for $\Omega \leq 0.4\Omega_0$ the disks are stable to the bar mode. Figure 6-2a, shows the bar shape spiral pattern for $\Omega = 0.8\Omega_0$ and does not for $\Omega = 0.4\Omega_0$.

Miller (1976) also looked at a Kalnajs disk, but with a polar grid code instead of a Cartesian grid. Figure 6-3 is taken from Miller (1976). It is the particle plot for a Kalnajs disk for $\Omega = 0.8\Omega_0$. The step-by-step structure of Figure 6-3 is different from that of Hohl's results (Figure 6-2a); however, a barlike structure appears within three rotations (instead of two as in Hohl's paper). The end result is that a Kalnajs disk of $\Omega = 0.8\Omega_0$ is susceptible to the m=2 bar mode instability in both a Cartesian and polar grid codes.

We should not be alarmed by the differences between Hohl (1972) and Miller (1976). Again we stress that exact results are not the issue; it is the global nature of the results that count. In both papers the bar mode is present and this is something that we can test. Hohl states that for $\Omega \leq 0.4\Omega_0$ the bar mode is not visible. Sellwood (1983), using a polar grid code, did extensive analysis on the Kalnajs disk $\Omega = 0.8\Omega_0$ model. His findings show that results can differ by as much as 50% in the growth rate of

instabilities, from using different methods to load the disk. Specifically Sellwood looked at two methods for loading the disk, a noisy and a quiet start. Sellwood defines a noisy start as a purely random load in both space and velocity. This is how Hohl loads his disks. A quiet start, as defined by Sellwood, evenly places the particles in tightly spaced rings (each ring is shifted slightly to prevent a spoked wheel effect). This is the load that Miller (1976) uses.

C) The Kalnajs Disk Load

a) Random Density Load

We have elected to use a random load in both space and velocity as did Hohl (1972). To randomly load a disk we define:

$\Sigma(r)$ = surface density at r,
Σ_0 = surface density at r = 0,
$M(r)$ = total mass enclosed within r,
R_{max} = radial edge of the disk.

We want to randomly place particles in x and y such that the density of particles is that of a Kalnajs disk. To accomplish this we use a random number to represent the fraction of the total mass of the disk. This will in turn define a radius at which we can place a particle. The ϕ component can then be determined by generating a random number between 0 and 2π. The polar coordinates r and ϕ can then be con-

verted to Cartesian coordinates x and y. This procedure is now demonstrated.

To determine a formula which can be used to randomly load a Kalnajs disk we start with the total mass enclosed in R:

$$M(R) = \int_0^{2\pi} \int_0^R \Sigma(r)\, r\, dr\, d\phi \quad . \tag{6-6}$$

We now define f to be the fraction of the total mass of the disk at radius R:

$$f = \frac{M(R)}{M_{tot}} \quad . \tag{6-7}$$

By definition $0 \leq f \leq 1$. Substituting in equation 6-6, we get:

$$f = \frac{2\pi \int_0^R \Sigma(r)\, r\, dr}{2\pi \int_0^{Rmax} \Sigma(r)\, r\, dr} \quad . \tag{6-8}$$

We now substitute in the surface density of the Kalnajs disk, equation 6-1. This gives:

$$f = \frac{\Sigma_o \int_0^R (1 - r^2/R_{max}^2)^{1/2}\, r\, dr}{\Sigma_o \int_0^{Rmax} (1 - r^2/R_{max}^2)^{1/2}\, r\, dr} \quad . \tag{6-9}$$

Note that the dependence on the quantity Σ_0 cancels out. Evaluating:

$$\int_0^R (1 - r^2 / R_{max}^2)^{1/2} r \, dr = {}^1/_{3Rmax} [R_{max}^3 - (R_{max}^2 - R^2)^{3/2}], \qquad (6\text{-}10)$$

and

$$\int_0^{Rmax} (1 - r^2 / R_{max}^2)^{1/2} r \, dr = \frac{R_{max}^2}{3} , \qquad (6\text{-}11)$$

and substituting these results into equation 6-9, we obtain:

$$f = \frac{{}^1/_{3\,Rmax} [R_{max}^3 - (R_{max}^2 - R^2)^{3/2}]}{R_{max}^2 / 3} . \qquad (6\text{-}12)$$

We now solve for R, and after some algebra we arrive at:

$$R = R_{max} [1 - (1 - f)^{2/3}]^{1/2} , \qquad (6\text{-}13)$$

where:

$$f = \text{random number } 0 \leq f < 1$$

Equation 6-13 is our random load equation; f is the random number, representing the fractional amount of total mass of the disk. If we define g to be a second random number, we can calculate the phi coordinate as:

$$\phi = 2\pi g . \qquad (6\text{-}14)$$

The actual x and y positions are then transformed by:

$$x = R\cos\phi \ , \ y = R\sin\phi \ . \qquad (6\text{-}15)$$

b) Verification of the density load

To verify that equations 6-13, and 6-14 satisfy the surface density of the Kalnajs disk (equation 6-1), we examine, with a separate program, the initial load made by using equations 6-13 and 6-14. This program reads and then bins the particles into annular rings of width one. The number in each bin is then divided by the area of that ring to give a surface density. Typically each bin contains two to three thousand particles. This result is then compared to the theoretical value of equation 6-1. Before this can be done, we must solve for Σ_0.

To solve for Σ_0, we start with the equation for total mass of the disk. From equation 6-6:

$$M_{tot} = 2\pi \int_0^{Rmax} \Sigma(r)\, r\, dr \ . \qquad (6\text{-}16)$$

Substituting in the surface density for a Kalnajs disk we get:

$$M_{tot} = 2\pi \Sigma_o \int_0^{Rmax} (1 - r^2/R_{max}^2)^{1/2}\, r\, dr \ . \qquad (6\text{-}17)$$

The integral in equation 6-17 has already been solved in

equation 6-11, so:

$$M_{tot} = 2\pi \Sigma_o R_{max}^2/3 \quad . \tag{6-18}$$

Solving for Σ_0 gives:

$$\Sigma_o = \frac{3 M_{tot}}{2\pi R_{max}^2} \quad . \tag{6-19}$$

For M_{tot} we use the total number of particles. This is satisfactory since for galactic simulations we set all particles to equal mass (equation 2-3). Figure 6-4 shows the points for the random load and a line for the theoretical surface density for a Kalnajs disk (note that the surface density is independent of omega). Agreement between the load and theory in Figure 6-4 is good. The deviations at the smaller radii are caused by low particle numbers in the annular bins. The good agreement in Figure 6-4 confirms the random density load as a particle representation of a Kalnajs disk.

c) Random Velocity Load

The velocities for a Kalnajs disk are governed by equation 6-3. The velocity dispersions are defined to be Gaussian (Kalnajs 1972). Therefore we use a random Gaussian number in both the radial and azimuthal velocity components. A random Gaussian number is a number chosen at random from a profile that has a Gaussian distribution centered about a

chosen value with a full-width-at-half-max equal to the dispersion calculated from equation 6-3. To load the velocities for a Kalnajs disk with a specific Ω and Ω_0, we first calculate the particle position from equations 6-13 and 6-14. Then we use equation 6-3 to calculate the correct velocity dispersion for both r and phi components. The specific radial and azimuthal velocities for the particle are assigned using the random Gaussian number generator with the calculated dispersion using v_r centered at zero and v_ϕ centered at the local disk angular velocity,

$$v_\phi = R\,\Omega \ . \qquad (6\text{-}20)$$

The velocities are then converted into Cartesian components by:

$$v_x = v_r \cos\phi - v_\phi \sin\phi \ , \qquad (6\text{-}21)$$

$$v_y = v_r \sin\phi + v_\phi \cos\phi \ . \qquad (6\text{-}22)$$

Figures 6-5 and 6-6 show the loaded radial velocity dispersions, compared to the theoretical values described by equation 6-3, for $\Omega = 0.4\Omega_0$ and $\Omega = 0.8\Omega_0$. Figures 6-5 and 6-6 confirm we have the correct random velocity distributions.

D) Results

a) Calculation of W_1

In chapters 4 and 5, we used dimensional analysis to calculate W_1. There is a short cut to this calculation which we use here. Squaring both sides of equation 6-2, we get:

$$2\,\Omega_0^2 R_{max} = \pi^2 G\,\Sigma_o \ . \qquad (6\text{-}23)$$

Solving for G:

$$G = \frac{2\,\Omega_0^2 R_{max}}{\pi^2 \Sigma_o} \ . \qquad (6\text{-}24)$$

Substituting equation 6-19 for Σ_0, we have the expression for G, or in program units W_1:

$$W_1 = G = \frac{4}{3}\,\frac{\Omega_0^2\, R_{max}^3}{\pi\ M_{tot}} \ . \qquad (6\text{-}25)$$

Since Ω_0 is the angular rotation rate for the rigid rotating Kalnajs disk, we use:

$$\Omega_0 = \frac{v_\phi(r = R_{max})}{R_{max}} \ , \qquad (6\text{-}26)$$

where:

$$v_\phi(r = R_{max}) = 2.0 \ . \qquad (6\text{-}27)$$

In chapters 4 and 5 we wanted the circular velocity to be less than one grid cell per time step to insure computational accuracy. However, if we were to use that here, the number of time steps for one disk revolution would be too large. For this reason we use a value of two. For the omega models we are using, the mean rotation rate is less than Ω_0 (the cold rotation rate). This means that the actual velocities are less than 2 grid cells per time step.

b) Initial Conditions for Omega Models

Throughout this thesis we have shown the effects of a softened potential on results. For tests having up to ten particles, we could afford to make many runs to illustrate the problems with a softened potential. For a disk of 102400 particles, this is no longer true. From examining previous results from Chapter 5, we have selected to use $\epsilon^2 = 5$ as our value for softening.

We will be comparing statistical output of the code CART2D with the statistical results of Hohl (1972). Hohl uses physical units to graph his results, while we use program units. Table 6-1 presents all the conversion factors necessary to compare results between the codes. For all the omega models used by Hohl, the total mass of each disk is 0.84×10^{11} solar masses. The outer edge of the disk is 16 kpc, and the cold disk rotation period is approximately

.41 Gyr (see Table 6-1). When applied to CART2D, each particle's mass is 0.82×10^6 solar masses, and one grid box length is 0.39 kpc. The way we have scaled the problem in equation 6-27, one time step is 3.2 Myr (see Table 6-1). In the next section, we convert a few cases to physical units to verify the results.

Hohl used a 128 X 128 cartesian grid. For our simulations we have kept the size of the disk the same in grid units, but have doubled the size of the grid to 256 X 256. This way the average number of particles per cell in the disk is the same as Hohl, but our space is twice as big. This reduces the number of grid spills we get for each run. A grid spill occurs when a particle exceeds the computational space, i.e., the particle crosses the boundaries of the grid. When this occurs, we simply remove the particle from the simulation. If a substantial number of particles spill, both total energy and angular momentum can be dramatically changed. To minimize this problem we have increased the grid size. All information concerning the omega model initial conditions are in Table 6-1.

c) Results for $\Omega = 0.4\Omega_0$

We ran the 0.4 omega model Kalnajs disk to 6 rotation periods, where there are 128.8 time steps per period (see Table 6-1). Figures 6-7 through 6-13 are the particle plots

for $\Omega = 0.4\Omega_0$. These can be directly compared to Figure 6-2a, Hohl's particle plots. Figure 6-7 shows the random spatial load. One period later, Figure 6-8 shows that while the outer regions of the disk have become more scattered than in the original load, the general appearance is similar. The last particle plot, Figure 6-13, is after six periods, and still shows a disk without any visible spiral structure. Figure 6-13 generally looks like all the previous figures suggesting that the disk is stable to the m = 2, or bar mode, in agreement with Hohl as seen in Figure 6-2a.

The radial velocity dispersions for this run are shown in Figures 6-14 through 6-20, covering zero to six rotational periods. The radial velocity dispersion is calculated by sorting the particles into rings of radial width equal to one grid unit. The radial velocity of each particle is then calculated as the dot product of the velocity with the radial position vector. In Cartesian coordinates this is:

$$\mathbf{v} \cdot \mathbf{r} = v_x x + v_y y \ , \tag{6-28}$$

and

$$v_r = \frac{\mathbf{v} \cdot \mathbf{r}}{|\mathbf{r}|} \ . \tag{6-29}$$

We then calculate the average radial velocity in each ring. CART2D uses the following expression to calculate the radial velocity dispersion:

$$\sigma_r^2 = \sum_{i=1}^{N} (v_r^2 - \bar{v}_r^2)/(N-1) \ , \qquad (6\text{-}30)$$

as described by Bevington (1969). Figure 6-14 shows the velocity dispersion for the load, which after making the conversions is what Hohl shows in Figure 6-2c. The conversion to physical units used by Hohl are as follows. The value for radial velocity dispersion at r = 0 is 1.0583 (see Figure 6-14). Multiplying by 0.39 kpc per grid and then converting kpc into km, dividing by $1x10^{14}$ (see Table 6-1) the number of seconds in one time step, we get 124.9 km/s. This agrees with Hohl's value to the precision available in Figure 6-2c. Hohl's velocity dispersion for the $\Omega = 0.4\Omega_0$ case flattens to an almost linear curve by the sixth period. We see the same trend to the velocity dispersions from CART2D, although our dispersions are less steep.

The surface density for the $\Omega = 0.4\Omega_0$ case is shown in Figures 6-21 through 6-27. Surface density is calculated by sorting the particles in radial rings (the same rings as for the velocity dispersion). The surface density is then equal to the total number of particles in each ring divided by the area of the ring. To compare with Figure 6-2d (Hohl's sur-

face density for the $\Omega = 0.4\Omega_0$ case), we simply multiply CART2D's surface density by the mass of a single particle (see Table 6-1) and divide by the square of the number of kpc in a grid (0.39 squared). Because the statistics are a little noisy at the center of the disk in Figure 6-21, we extrapolate the curve to a value of 28 for r equaling zero. Converting to physical units gives 1.5×10^8 solar masses per square kpc. This is very close to what Figure 6-2d shows for t=0, Hohl's initial load. Comparing the time evolution of the surface density shows our disk becoming a little more dense at the center than Hohl's, but this is not a big concern because the total number of particles in the innermost rings is small compared to the other rings. We can see by the second period that the surface density basically remains the same out to six periods.

The Q value used by Hohl in Figure 6-2b, is a measure of the stability of a system to axisymmetric modes. In theory, Q values refer to the global stability of a system. However here (as in Hohl 1972), we use the Q values for local measures of stability. From Binney and Tremaine (1989) (originally from Toomre 1964):

$$Q = \frac{\sigma_r(r)\,\kappa(r)}{3.36\,G\,\Sigma(r)}\,, \qquad (6\text{-}31)$$

where:

$\sigma_r(r)$ = radial velocity dispersion at r,
$\kappa(r)$ = epicyclic frequency at r,
G = gravitational constant,
$\Sigma(r)$ = surface density at r.

We already have the radial velocity dispersion and surface density from sorting particles into annular rings. The epicyclic frequency is calculated as (Hockney and Eastwood 1988):

$$\kappa^2 = \frac{-3F(r)}{r} - \frac{dF(r)}{dr}, \qquad (6\text{-}32)$$

where F(r) is the force at r. The force values are calculated in the code by equations 2-9 and 2-10. To make the calculation in equation 6-32 simpler we take the derivative of the force only along the X and Y axes of the grid at a given value of r. This gives four separate values of the epicyclic frequency for a given radial ring. These four values are then averaged for use in equation 6-32. Unfortunately Hohl (1972) does not state how he evaluates his epicyclic frequency. This is of some concern when comparing Q values. Figures 6-28 to 6-34 show the Q values for the $\Omega = 0.4\Omega_0$ case. Since Q is dimensionless we do not need to make any unit conversions as we did in comparing the velocity dispersion and surface density. Figure 6-28 shows the Q values for the load. This is what Hohl has in

Figure 6-2b. This means that for the axisymmetric case, our four point value for the epicyclic frequency and our calculation of Q matches Hohl (1972). In fact Figures 6-29 to 6-34 agree rather well with Hohl's results in Figure 6-2b. The slight rise towards the edge of the disk in both Hohl's results and ours is mostly due to low particle numbers (100 to 200 particles per ring) in the outer regions.

Overall the results we obtain for the Kalnajs disk 0.4 omega model agree rather well with Hohl's (1972) results. Figure 6-35 shows the percentage of the disk that spilled as a function of time step. Increasing the grid space certainly helped as Figure 6-35 shows only 6% or so of the disk has spilled. It is also important to notice in Figures 6-7 to 6-13 that there is no visible bar mode instability. If the bar mode were present it would indicate an error in some aspect of the load or the code itself. The fact that it is not seen and that our statistics agree with Hohl (1972) shows that CART2D is working as expected.

c) Results for $\Omega = 0.8\Omega_0$

The omega 0.8 model uses the same initial conditions as the 0.4 model, except the velocity dispersion is not as high (see Figure 6-6). Since we have the same initial conditions, one period is 128.8 time steps and we use the same 256 X 256 grid. We present results for the omega 0.8 case

out to four rotation periods (as opposed to six in the omega 0.4 case). Figures 6-36 through 6-40 are the particle plots for the omega 0.8 model. These can be directly compared to the $\Omega = 0.8\Omega_0$ model in Figure 6-2a. Figure 6-37 is one period into the simulation and there appears to be no noticeable difference from the results for the omega 0.4 model. However by two periods, Figure 6-38 shows a disk which is not round in shape (the m=2 bar mode is starting to show). By the third period, the bar mode instability is easily seen as a barred spiral pattern in Figure 6-39. By the fourth period, the disk has collapsed and scattered particles everywhere. Figure 6-40 shows the collapsed disk. When comparing to the omega 0.8 case of Hohl in Figure 6-2a, our results seem to be about one period behind. This is not unusual in light of Sellwood's (1983) comment that results can differ from run to run by as much as 50% in the growth of instabilities. The results of Miller (1976) (Figure 6-3) show a bar like structure happening by the third rotational period. Figures 6-36 to 6-40 show the expected bar mode instability found by Hohl and Miller, thus confirming what we originally wanted to show, a bar like spiral developing for the omega 0.8 model.

The radial velocity dispersion is calculated in the same manner as for the omega 0.4 case. Figures 6-41 through 6-45 show the velocity dispersion for the omega 0.8 model.

After converting to physical units, these charts compare well with the velocity dispersions in Figure 6-2c. From an examination of these figures (6-41 to 6-45), we see that our dispersion becomes flatter than Hohl's results, but otherwise agrees. Our results for period three (comparing to period two of Hohl) do not possess as much structure but do have a rather flat region in dispersion for the inner part of the disk.

The surface density for the omega 0.8 model is shown in Figures 6-46 through 6-50. These plots show that the surface density increases dramatically towards the center. This matches well with Hohl's results in Figure 6-2d.

The Q values for the omega 0.8 case appear to be the only disagreement with Hohl's results. Figures 6-51 through 6-55 show our Q values calculated as before in the omega 0.4 case. Hohl's Q values in Figure 6-2b show quite a bit of structure for the bar case and beyond. Our Q values show less variation than Hohl's. The initial Q values match up exactly (Figure 6-51), but we do not see any real structure until the fourth period. Since the velocity dispersion and surface density all match rather well, we deduce that the calculation for the epicyclic frequency may be a problem. Since we are using only four points on the principal axes, we believe that there are not enough points to correctly evaluate the epicyclic frequency in cases of severe

nonaxisymmetry. For example, examine Figure 6-54 for period three. We can see from the particle plot (Figure 6-39) that the bar lies more along the X axis than the Y. Therefore we could have large values for the epicyclic frequency there, and small values along the Y axis. The average would suffer because there are not enough points to get a good statistical base.

With the exception of the Q values the Kalnajs disk omega 0.8 model compares well with the results of Hohl (1972). Our time development appears to be slower by one period, but Miller (1976) also shows bar structure by the third period so this delay is not a concern. The omega 0.8 model does show the expected bar mode instability for which we were looking, thus verifying the runs. Figure 6-56 shows the percent of the disk that has spilled during the run. As can be seen from this figure, by the fourth period 15% of the disk has spilled out over the edge of the grid. This is of concern because the disk is losing angular momentum and energy with each spill. After the fourth period the spills continue to escalate and the simulation is stopped.

E) Summary of Kalnajs Disk Tests

Both Kalnajs disk runs compare well with the results of Hohl (1972). The particle plots for the omega 0.8 model also compare (generally) with the polar results of Miller

(1976). From these comparisons we conclude that CART2D functions as other N-body codes, and thus is an appropriate tool for N-body research.

CHAPTER 7

Summary and Concluding Remarks

The purpose of this thesis is not only to show that the n-body FFT grid code CART2D functioned as expected, but also to outline a series of results that can be used to verify other 2-dimensional n-body codes. We have accomplished both goals in the following way. First, showing the performance of CART2D in four tests: the single particle potential test, the two body problem, the ten body problem, and the Kalnajs disk test. Second, we outlined a method of approach in validation of n-body codes by starting simple single particle tests, and ending with complex 102,400 particle disks. Each test is described in detail with initial conditions, calculations necessary to start each run, and final results. We have brought together published n-body results for a Kalnajs disk, and shown how CART2D results compare. All of these items, together accomplished the purpose of this work.

WORKS CITED

Bevington, Philip R., Data Reduction and Error Analysis for the Physical Sciences. New York: McGraw-Hill, 1969

Binney, James, and Scott Tremaine. Galactic Dynamics. Princeton: Princeton University Press, 1987.

Cromer, Alan. "Stable Solutions Using the Euler Approximation." American Journal of Physics 49 (1981): 455-459.

Danby, J. M. A. Fundamentals of Celestial Mechanics. Richmond: Willmann-Bell, Inc., 1988.

Hockney, R. W., and J. W. Eastwood. Computer Simulation Using Particles. Bristol: Adam Hilger, 1988.

Hohl, Frank. "Evolution of a Stationary Disk of Stars." Journal of Computational Physics 9 (1972): 10-25.

Kalnajs, A. J. "The Equilibria and Oscillations of a Family of Uniformly Rotating Stellar Disks." The Astrophysical Journal 175 (1972): 63-76.

Mihalas, D. Galactic Astronomy. San Francisco: W. H. Freeman and Company, 1968.

Miller, R. H., and K. H. Prendergast. "Stellar Dynamics in a Discrete Phase Space." The Astrophysical Journal 151 (1968): 699.

Miller, R. H. "Validity of Disk Galaxy Simulations." Journal of Computational Physics 21 (1976): 400-437.

Sellwood, J. A. "Quiet Starts for Galaxy Simulations." Journal of Computational Physics 50 (1983): 337-359.

---. The Art of N-Body Building. Annual Reviews of Astronomy and Astrophysics, no. 25. Palo Alto: Annual Reviews Inc, 1987.

Toomre, A. "On the Gravitational Stability of a Disk of Stars." The Astrophysical Journal 139 (1964): 1217.

TABLES

Table 4-1

Conservation of Angular Momentum

ASQ	Average	Standard Deviation	% of Norm
no mass	47.6563	1.275×10^{-3}	2.676×10^{-3}
30.00	47.6362	7.604×10^{-3}	1.596×10^{-2}
2.00	49.2767	1.0572	2.1454
1.00	48.2111	1.3045	2.7057
0.75	48.5481	1.1858	2.4425
0.50	44.4360	1.7484	3.9346
0.25	-8.2634	28.799	348.52

Table 4-2

Conservation of Energy

ASQ	Average	Standard Deviation	% of Norm
no mass	-0.2772	2.446×10^{-3}	0.8825
30.00	-0.2839	2.509×10^{-3}	0.8838
2.00	-0.2815	1.133×10^{-2}	4.0247
1.00	-0.3058	1.511×10^{-2}	4.9391
0.75	-0.3066	1.372×10^{-2}	4.4759
0.50	-0.3631	2.125×10^{-2}	5.8525
0.25	-0.3995	1.778×10^{-2}	4.4508

Table 5-1

Initial Conditions, 10 Body Test

Particle #	Radius	X	Y	Period
0	0.0	128.5	128.5	Core Mass
1	10.0	138.5	128.5	83.77
2	20.0	128.5	148.5	236.95
3	30.0	98.5	128.5	435.31
4	40.0	128.5	88.5	670.20
5	50.0	178.5	128.5	936.64
6	60.0	128.5	188.5	1231.24
7	70.0	58.5	128.5	1551.54
8	80.0	128.5	48.5	1895.63
9	90.0	218.5	128.5	2261.94

Table 5-2

Conservation of Angular Momentum
10 Body Test

ASQ	Average	Standard Deviation	% of Norm
30.00	144.7499	0.0169	0.0117
5.00	144.5517	0.1624	0.1123
2.00	144.7402	0.1842	0.1273
1.00	145.5144	0.8006	0.5502
0.75	145.5342	1.4475	0.9946
0.50	132.8892	8.3226	6.2628
0.25	124.2391	13.4848	10.8539

Table 6-1

Kalnajs Disk Conversion Factors

Description	Item	Program Units	Physical Units
Total Mass of Disk	M_{tot}	102400	8.4×10^{10} M_s
Mass of Particle	m_{part}	1	8.2×10^{5} M_s
Outer Radius of Disk	R_{max}	41.0	16.0 kpc
Grid to kpc	z	1 grid	0.390 kpc
Grid Size	L x L	256 x 256	99.84 x 99.84 kpc
Period of Cold Disk	Γ	128.8	416.15 Myr
Time Step	Δt	1	3.23087 Myr
Cold Disk Angular Rate	Ω_0	4.878048×10^{-2}	1.5×10^{-2} Myr
Gravitational Constant	W_1	6.797242×10^{-4}	4.656×10^{-12} $\frac{kpc^3}{M_s\ Myr^2}$
Conversions:			
solar mass		1 M_s	1.9891×10^{30} kg
kilo-parsec		1 kpc	3.085678×10^{16} km
year		1 yr	3.155328×10^{7} sec

FIGURES

FIGURE 2-1A

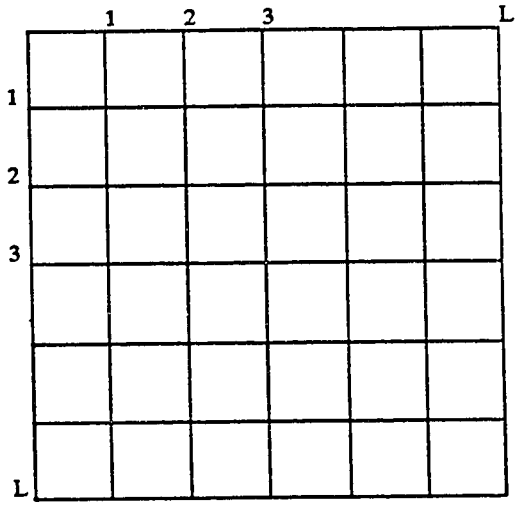

Fig 2-1a: L x L Cartesian Potential Grid

FIGURE 2-1B

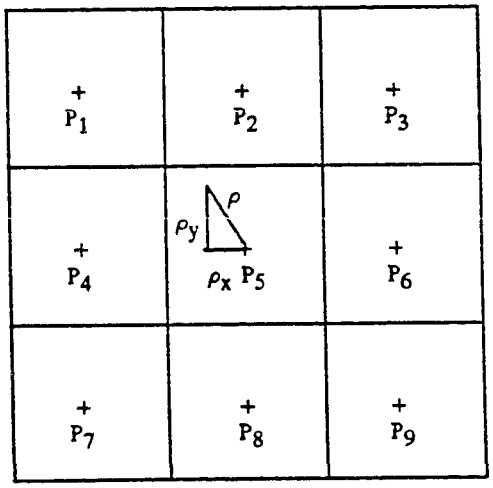

Fig 2-1b: Particle in cell with nearest neighbors.
ρ_x = x-component of particle's distance from cell center
ρ_y = y-component of particle's distance from cell center
P_i = the potential in cell i

FIGURE 2–2
IMAGE FORCE

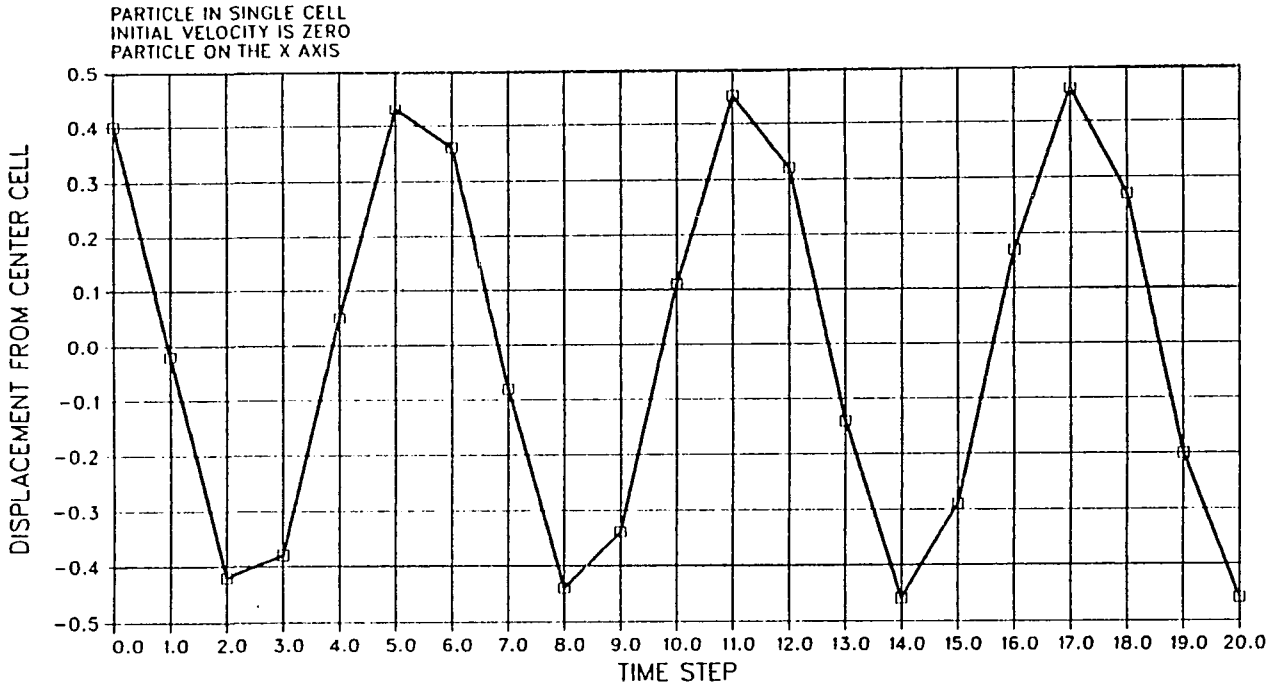

Fig 2-2: Particle is in a single cell, displaced 0.4 from the center along the x-axis. The particle initially is at rest, and there are no external forces present. Graph shows harmonic motion of particle along the x-axis as a function of time.

FIGURE 2-3
IMAGE FORCE

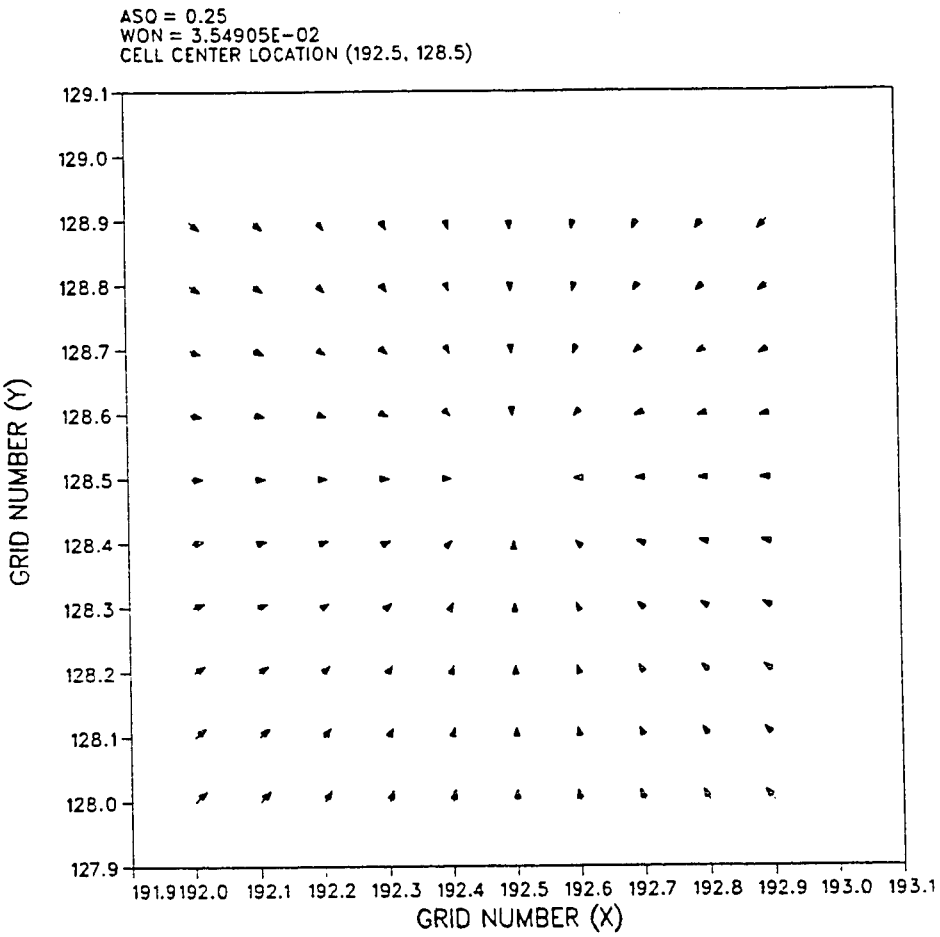

Fig 2-3: Figure shows vector force field of the image force for a single cell. Note that all force vectors point to the center of the cell. There are no external forces present.

FIGURE 3-1

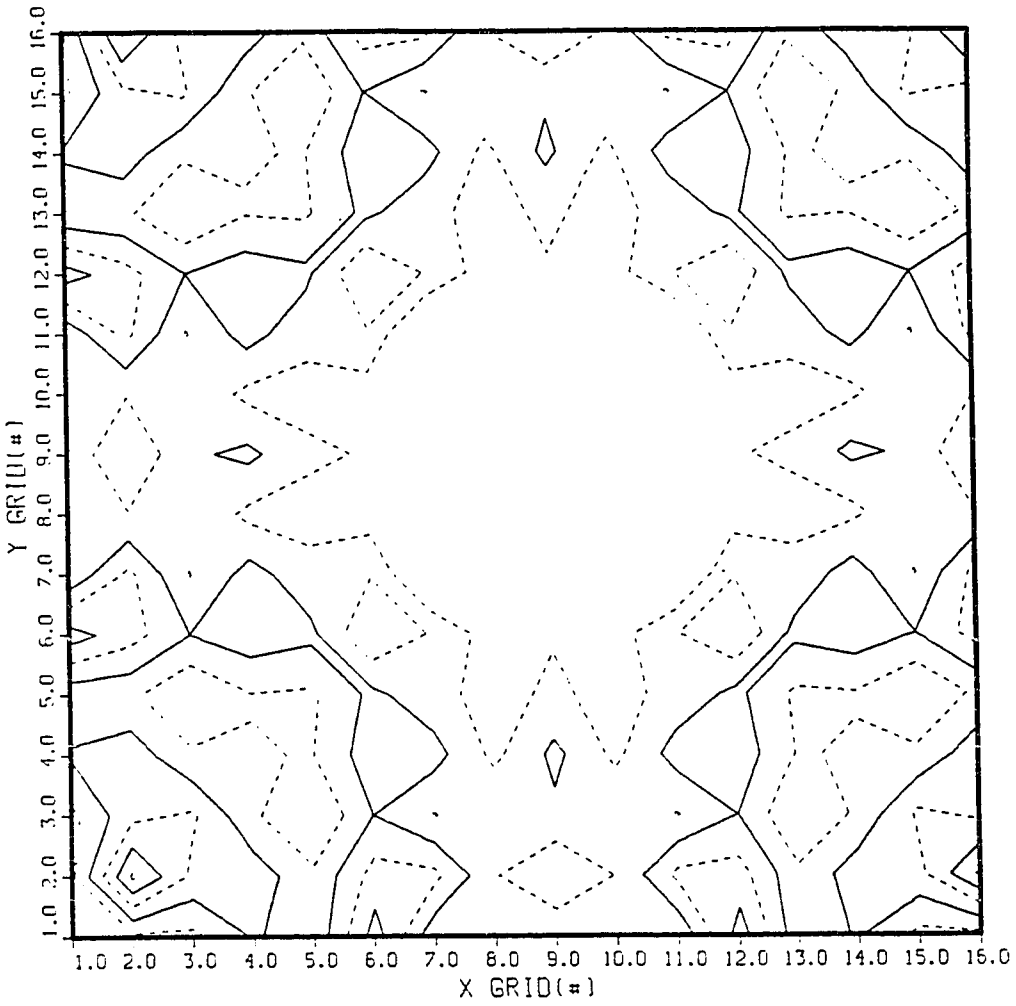

Fig 3-1: Contour graph of the percent difference between the analytic potential and the calculated potential, for a single particle located at the center of the grid (8.5, 8.5). The contour increment is 1.0×10^{-5} percent difference on a 16 X 16 grid.

FIGURE 3-2

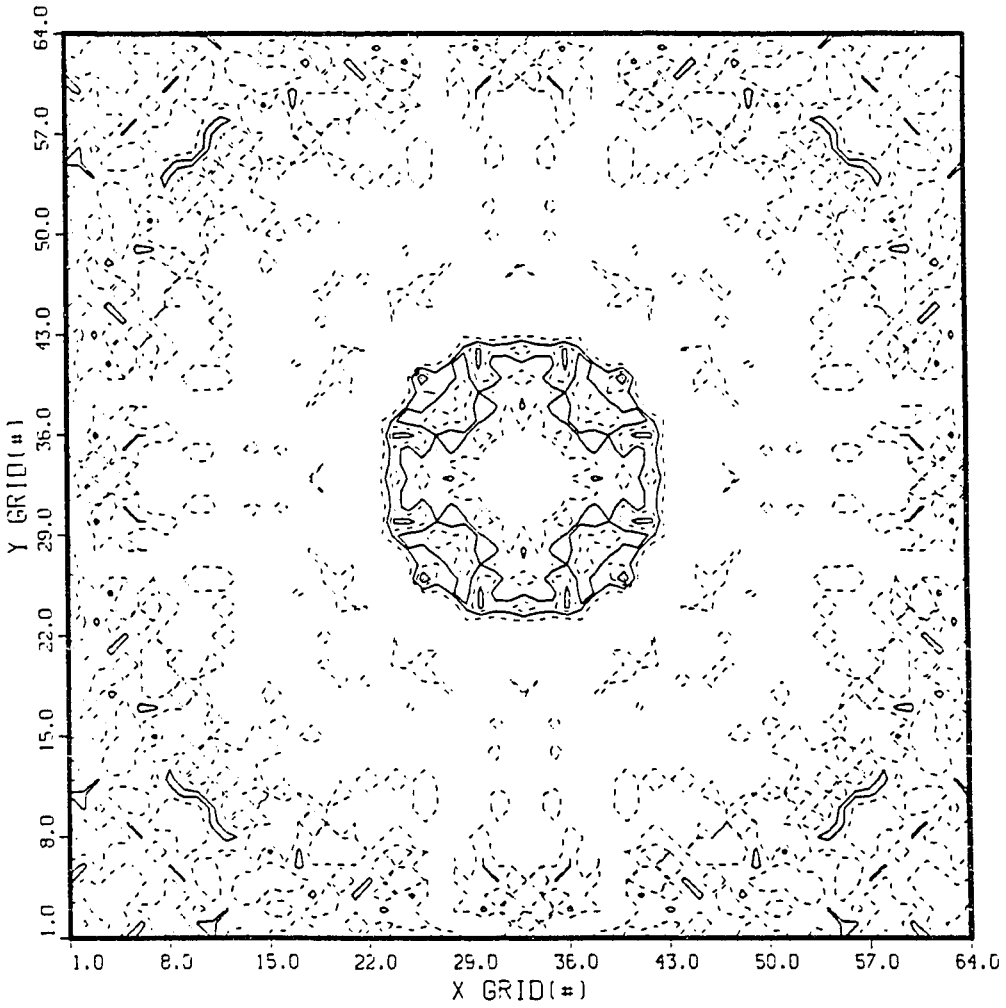

Fig 3-2: Contour graph of the percent difference between the analytic potential and the calculated potential, for a single particle located at the center of the grid (32.5, 32.5). The contour increment is 1.0×10^{-5} percent difference on a 64 X 64 grid.

FIGURE 4–1
PARTICLE WITH MASS REMOVED

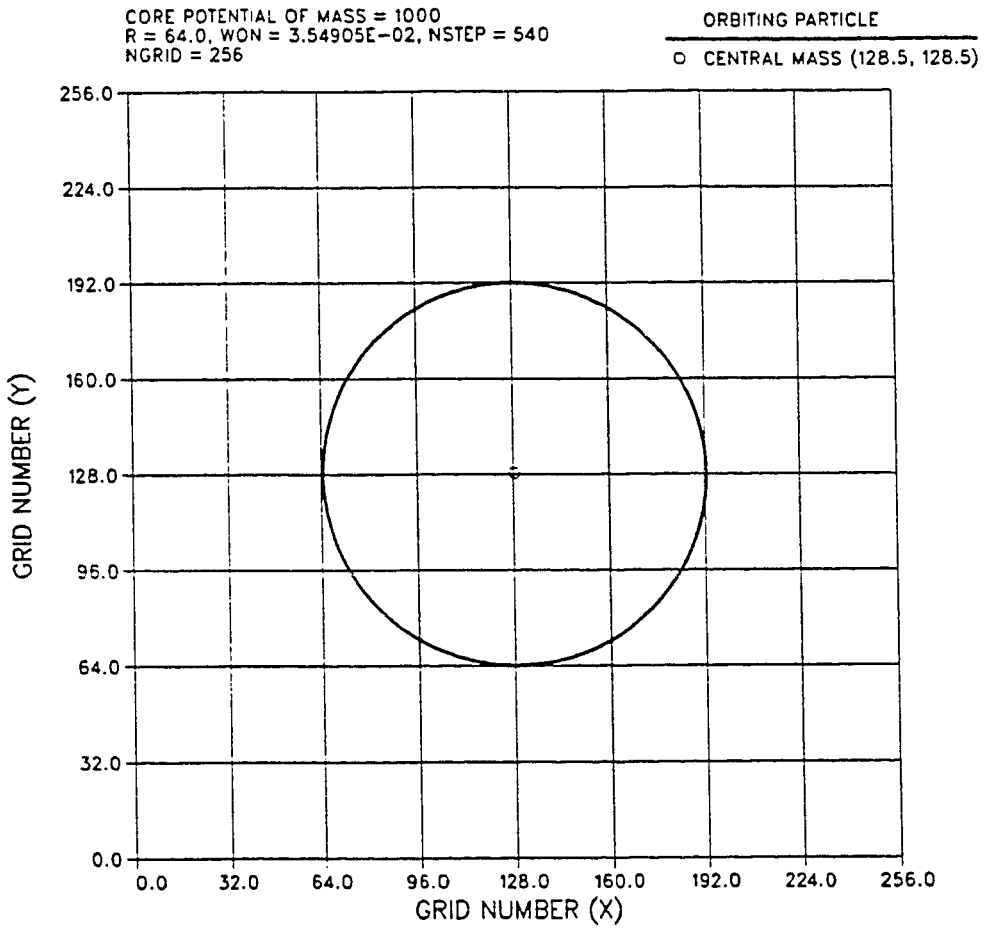

Fig 4-1: Particle is in a circular orbit about a core mass equivalent to 1000 particles. Particle mass is removed from the potenital calculation to avoid effects from the image force. Initial radius is 64, on a 256 X 256 grid. The orbital period is 540 time steps.

FIGURE 4–2
PERCENT ERROR IN ORBITAL PATH

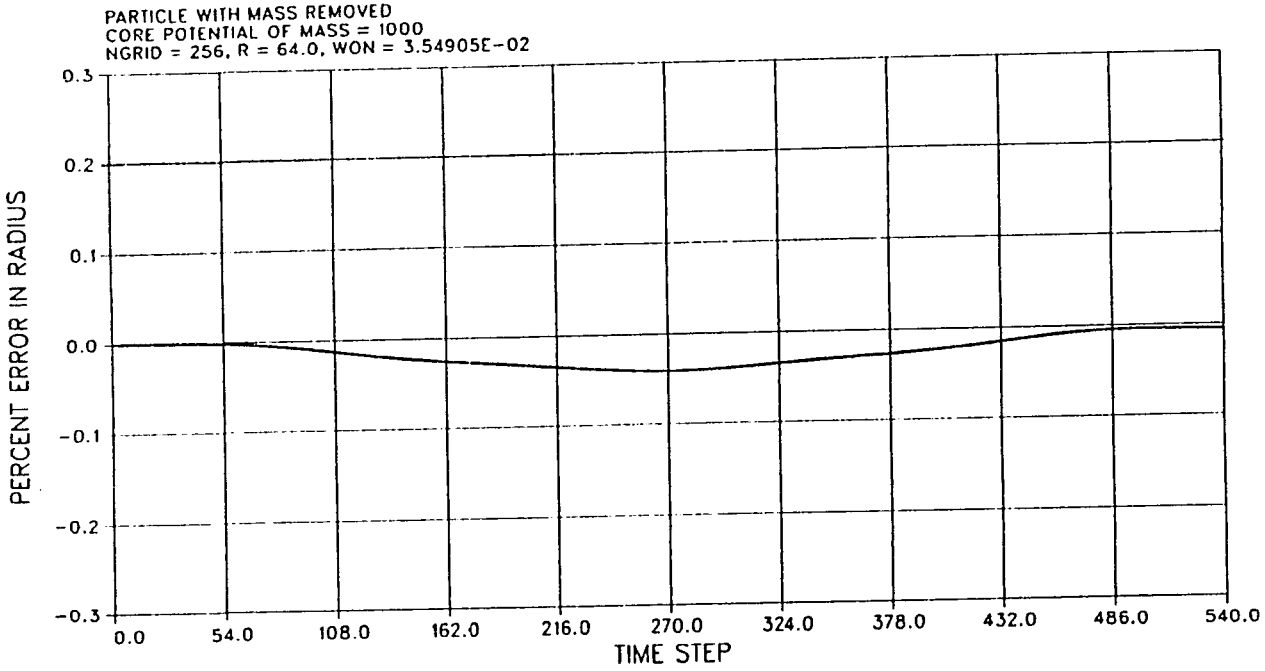

Fig 4-2: Percent error in the orbital path for the particle in Figure 4-1.

FIGURE 4–3
ASQ = 2.00

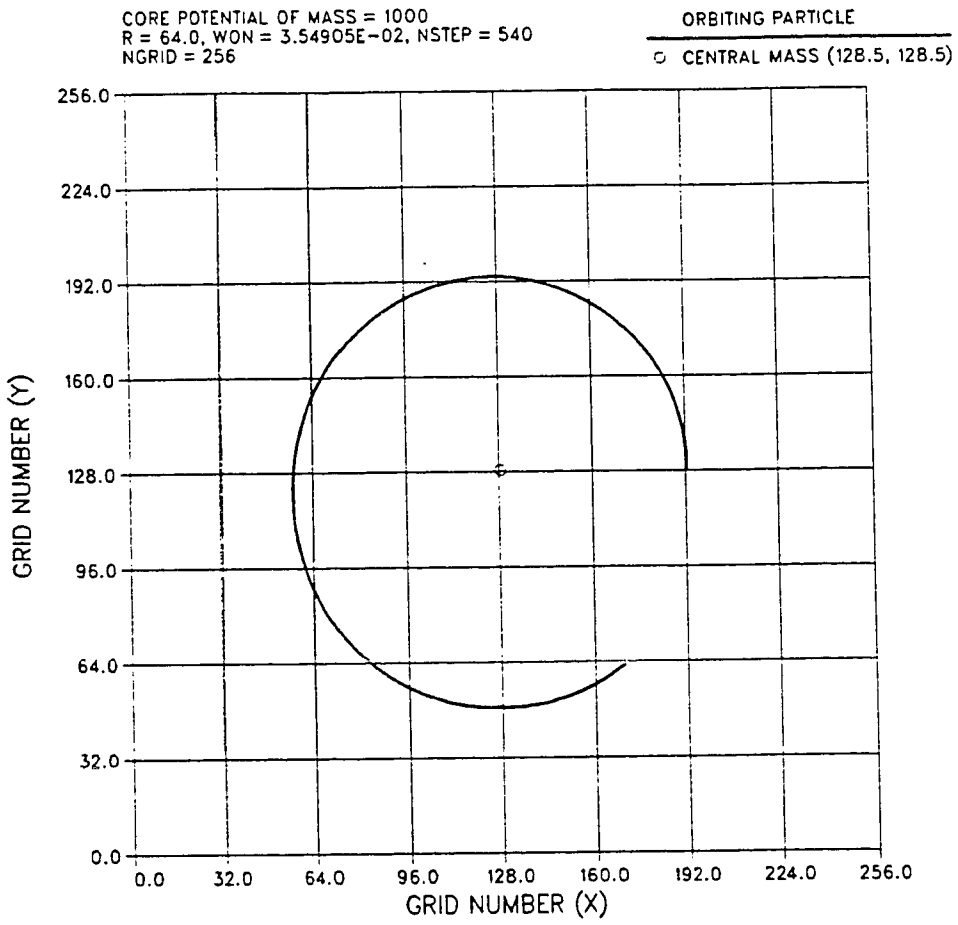

Fig 4-3: Particle in orbit about a massive core with a square of the softening parameter equal to two (ASQ = ϵ^2 = 2.00). Core mass is repressented by an effective potential equivalent to 1000 particles located at the center of the grid. Initial radius is 64, on a 256 X 256 grid. This Figure plots 540 time steps.

FIGURE 4–4
ASQ = 1.00

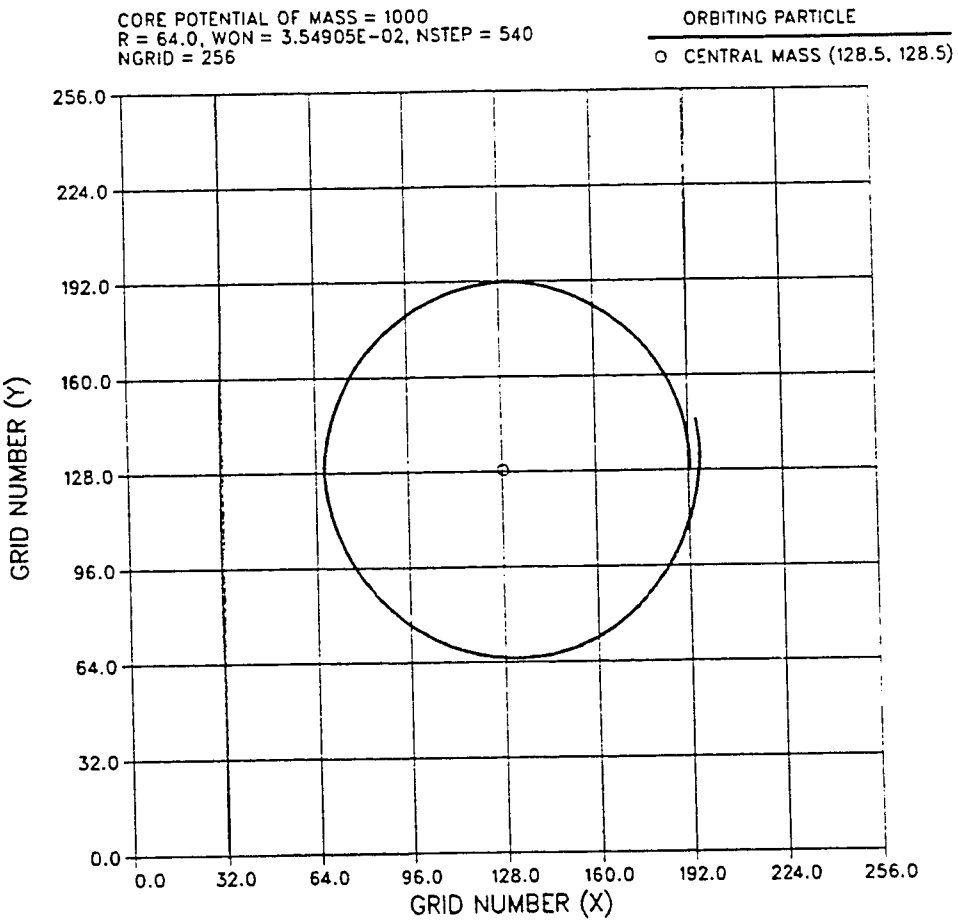

Fig 4-4: Particle in orbit about a massive core with a square of the softening parameter equal to one (ASQ = ϵ^2 = 1.00). Core mass is repressented by an effective potential equivalent to 1000 particles located at the center of the grid. Initial radius is 64, on a 256 X 256 grid. This Figure plots 540 time steps.

FIGURE 4–5
ASQ = 0.75

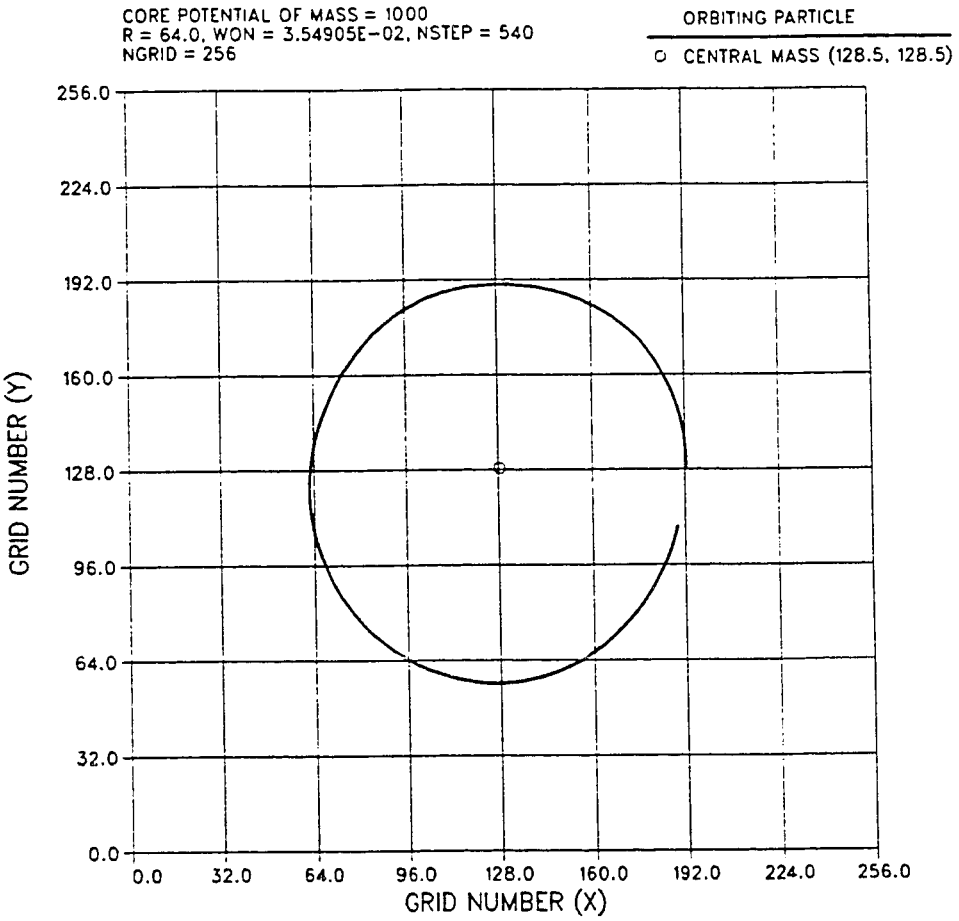

Fig 4-5: Particle in orbit about a massive core with a square of the softening parameter equal to 0.75 (ASQ = ϵ^2 = 0.75). Core mass is repressented by an effective potential equivalent to 1000 particles located at the center of the grid. Initial radius is 64, on a 256 X 256 grid. This Figure plots 540 time steps.

FIGURE 4–6
ASQ = 0.50

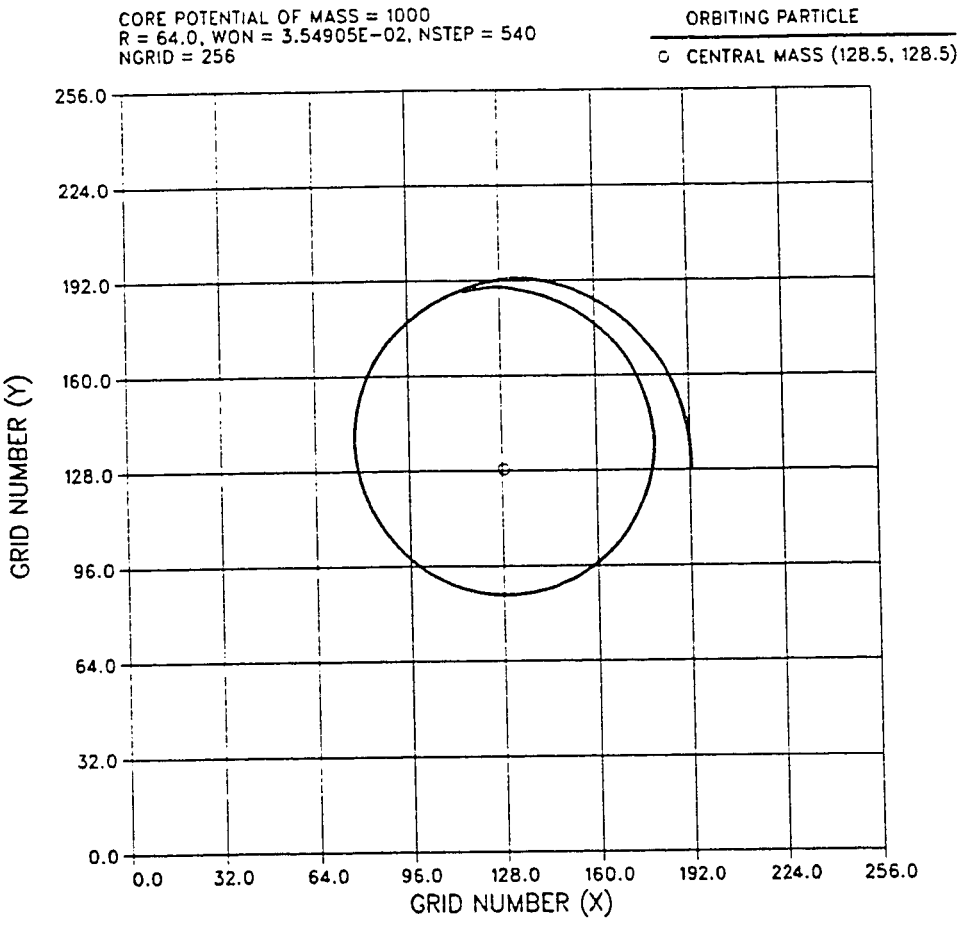

Fig 4-6: Particle in orbit about a massive core with a square of the softening parameter equal to 0.50 (ASQ = ϵ^2 = 0.50). Core mass is repressented by an effective potential equivalent to 1000 particles located at the center of the grid. Initial radius is 64, on a 256 X 256 grid. This Figure plots 540 time steps.

FIGURE 4-7
ASQ = 0.25

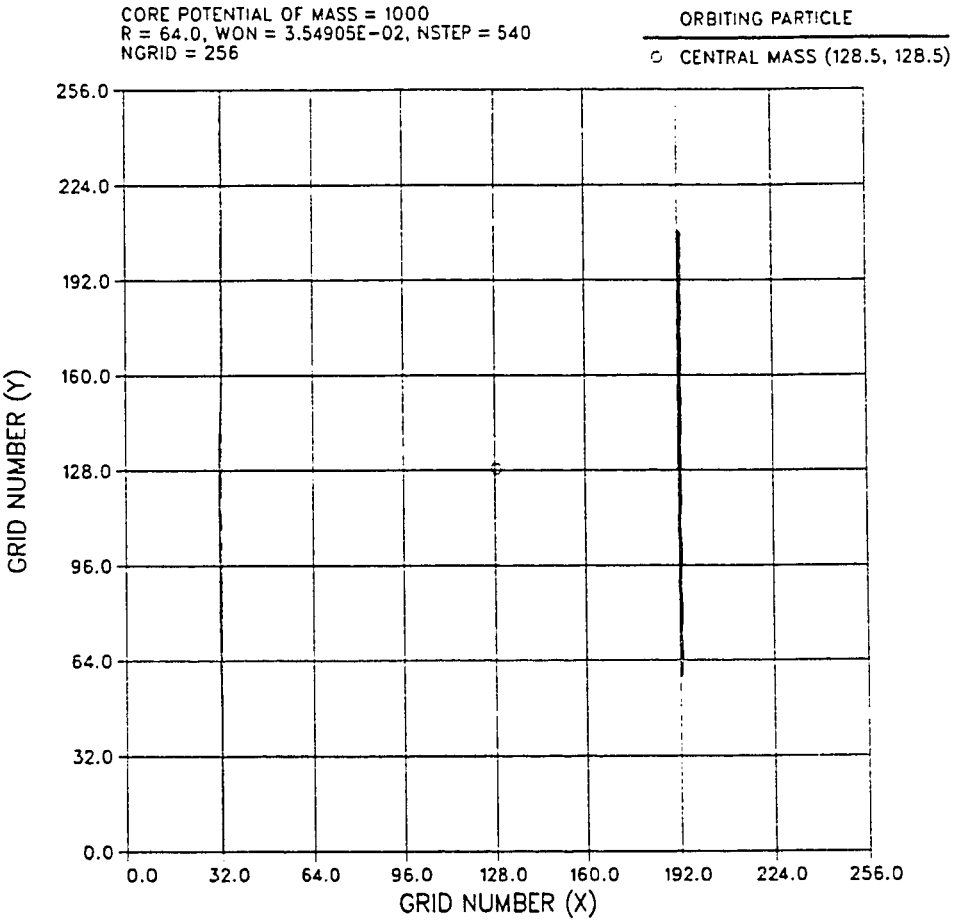

Fig 4-7: Particle in orbit about a massive core with a square of the softening parameter equal to 0.25 (ASQ = ϵ^2 = 0.25). Core mass is repressented by an effective potential equivalent to 1000 particles located at the center of the grid. Initial radius is 64, on a 256 X 256 grid. This Figure plots 540 time steps.

FIGURE 4–8
PERCENT ERROR IN ORBITAL PATH

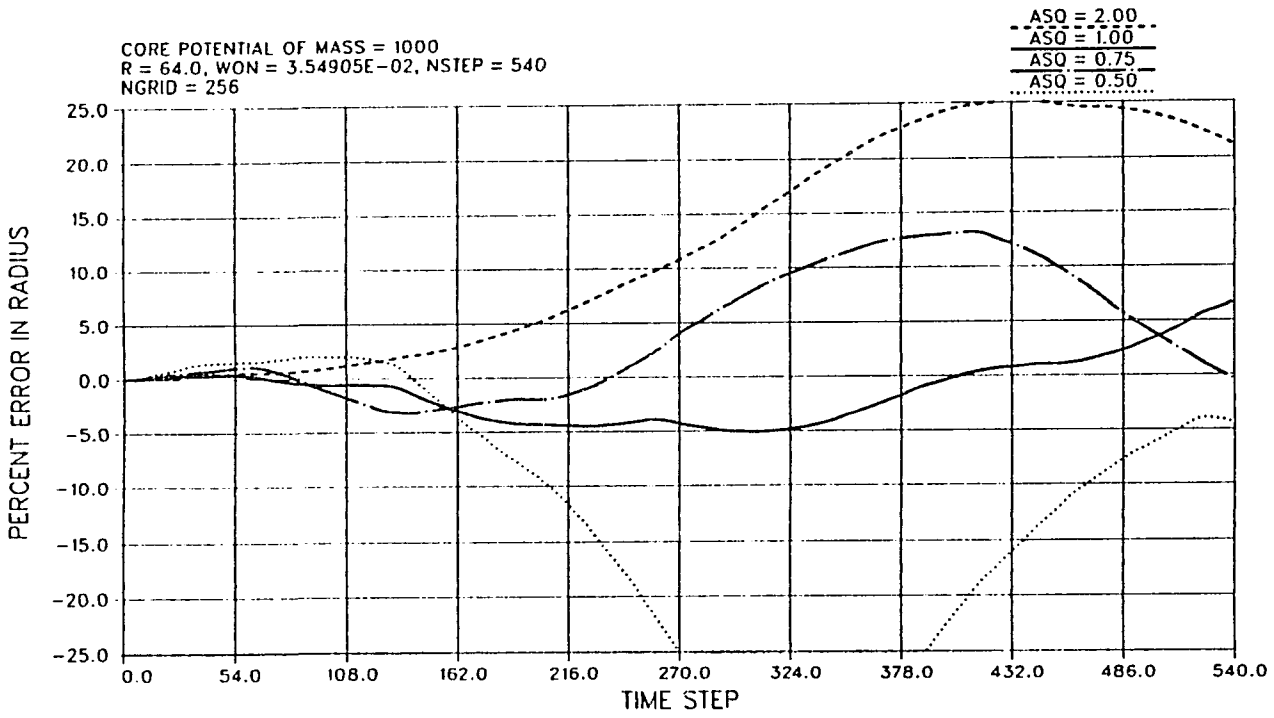

Fig 4-8: Percent eror in orbital path for a particle in circular orbit for ASQ = ϵ^2 = 2.00, 1.00, 0.75, and 0.50. Each curve corresponds to the orbital path in figure 4-3, 4-4, 4-5, and 4-6.

FIGURE 4-9
SOFTENING OF ASQ = 2.00

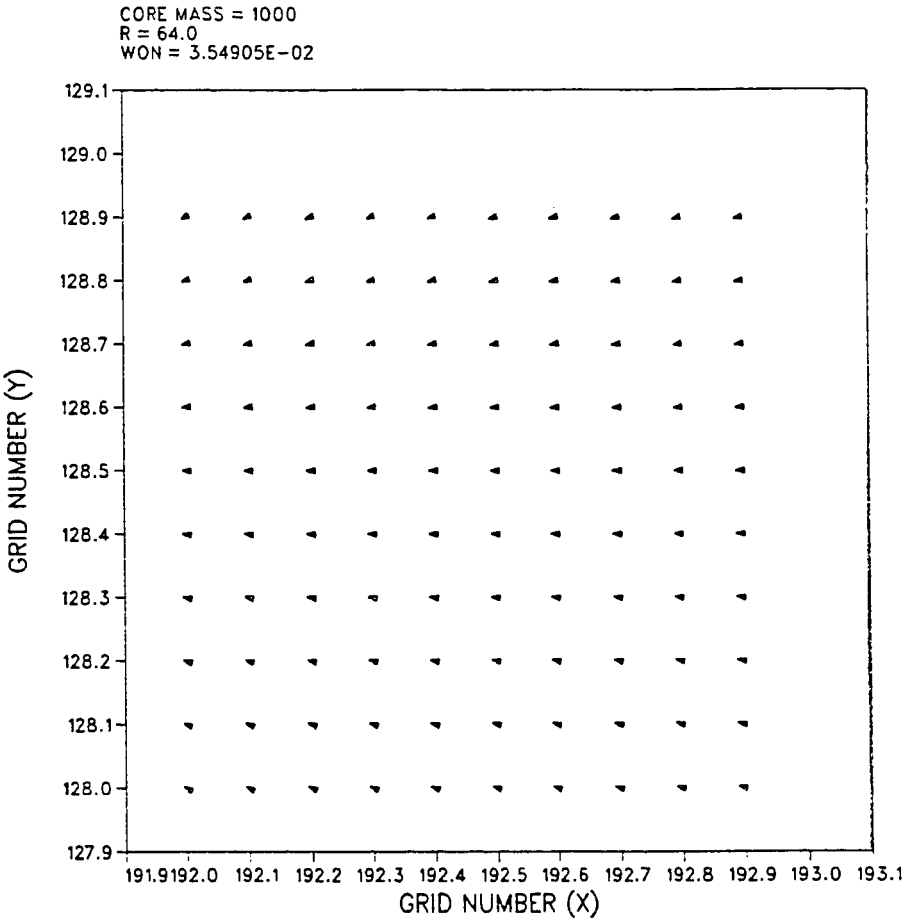

Fig 4-9: Vector force field of a single cell for particle in Figure 4-3. Core mass is located at (128.5, 128.5) which is to the left in this figure.

FIGURE 4-10
SOFTENING OF ASQ = 1.00

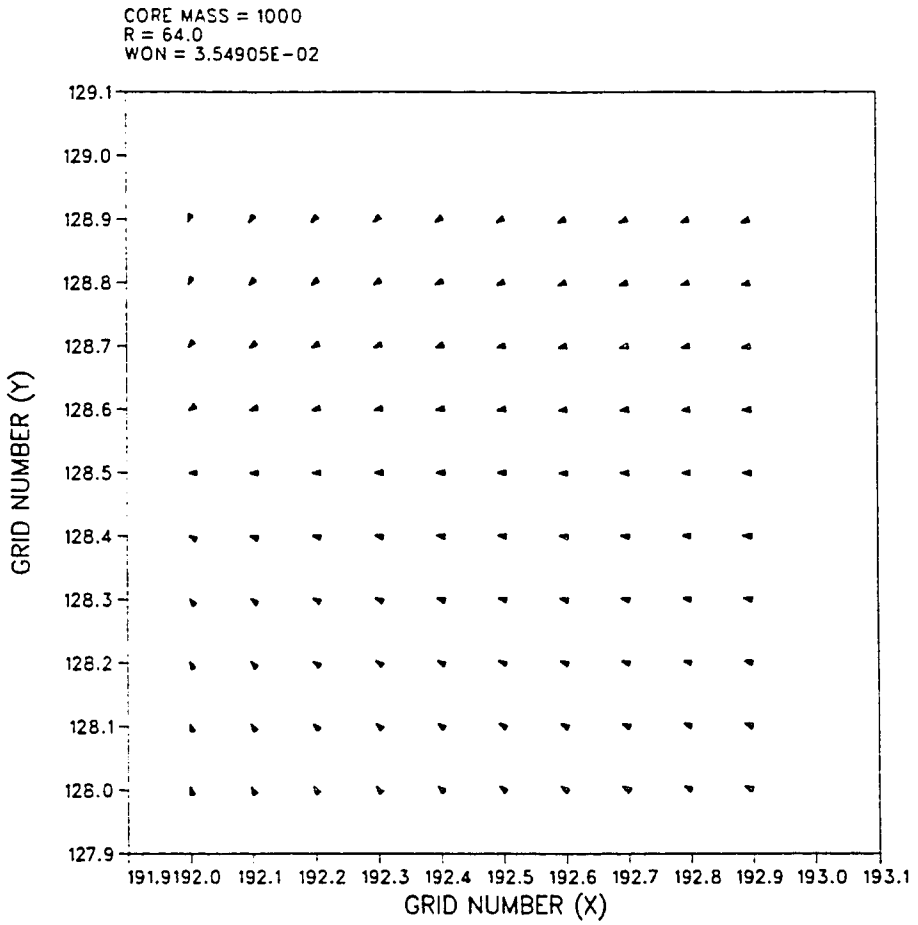

Fig 4-10: Vector force field of a single cell for particle in Figure 4-4. Core mass is located at (128.5, 128.5) which is to the left in this figure.

FIGURE 4–11
SOFTENING OF ASQ = 0.75

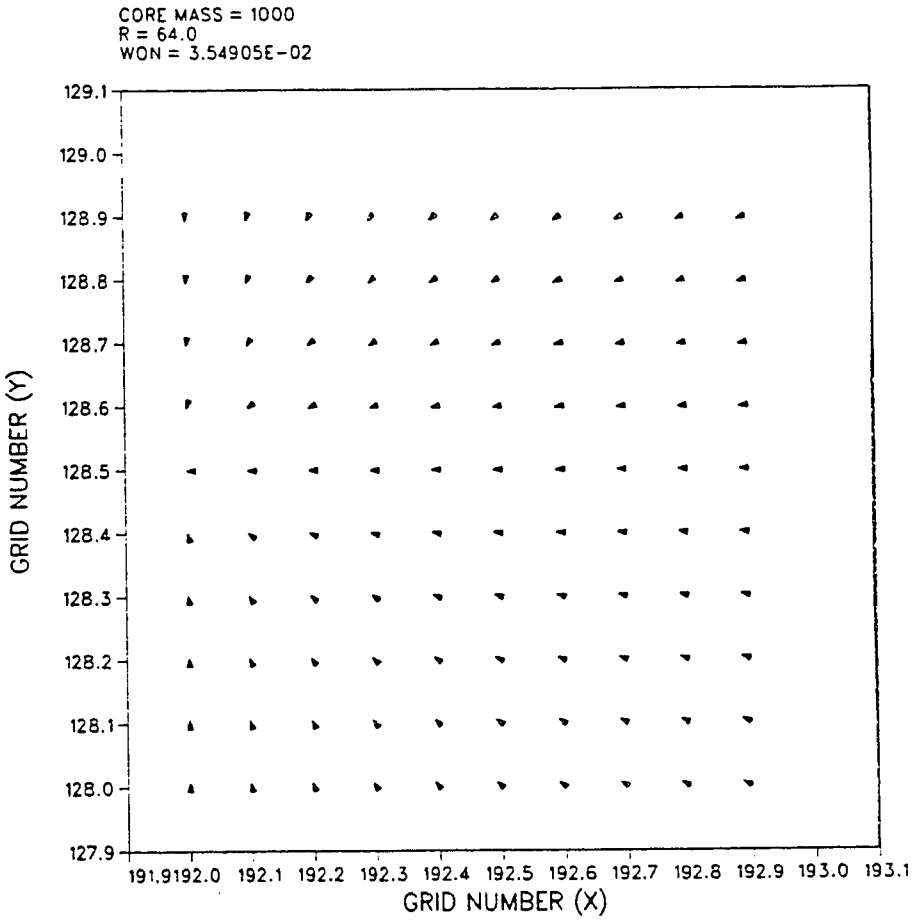

Fig 4-11: Vector force field of a single cell for particle in Figure 4-5. Core mass is located at (128.5, 128.5) which is to the left in this figure.

FIGURE 4–12
SOFTENING OF ASQ = 0.50

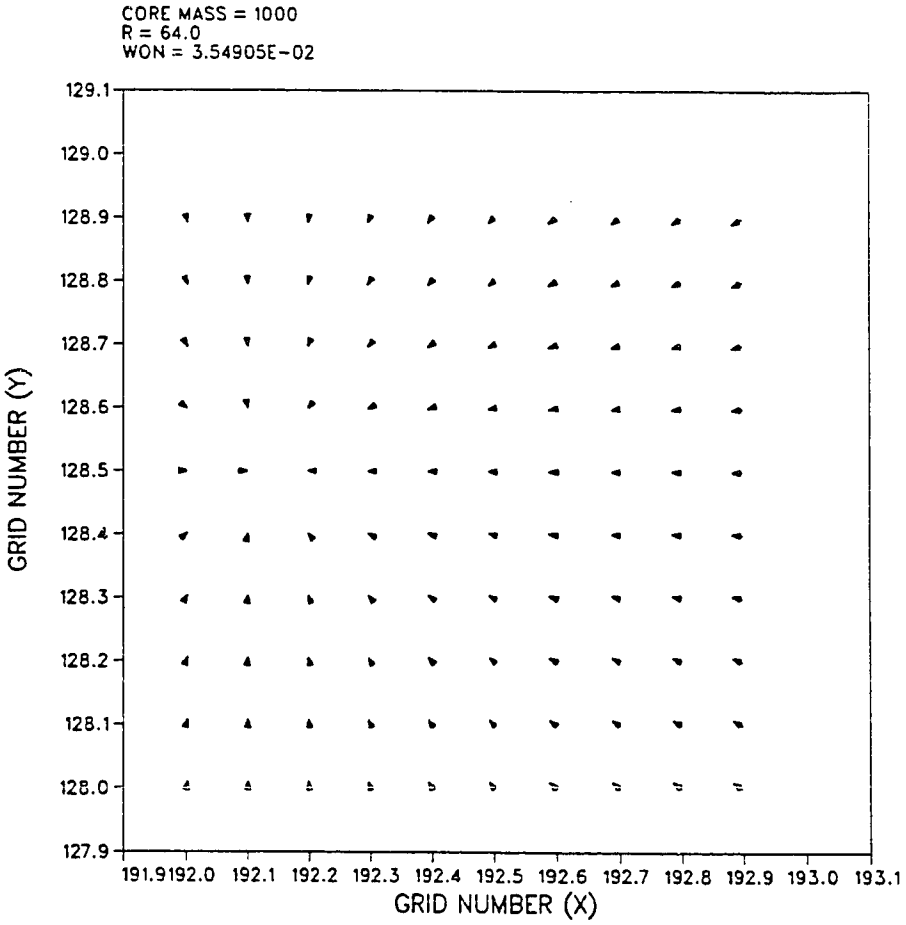

Fig 4-12: Vector force field of a single cell for particle in Figure 4-6. Core mass is located at (128.5, 128.5) which is to the left in this figure.

FIGURE 4–13
SOFTENING OF ASQ = 0.25

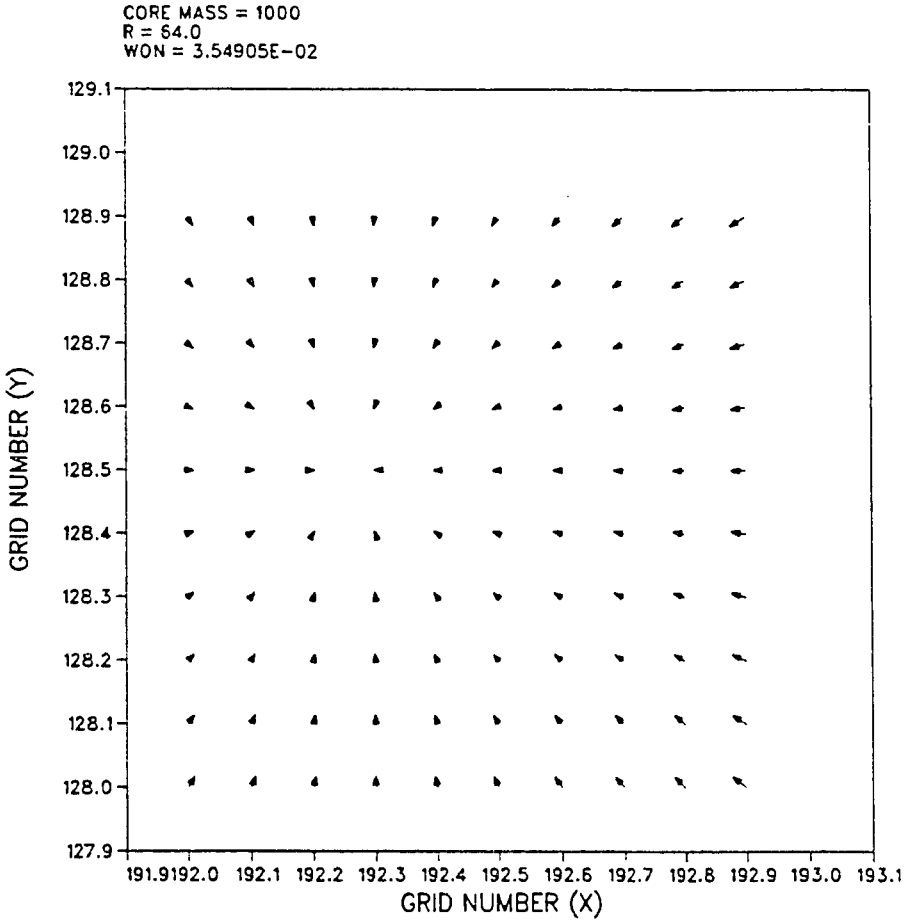

Fig 4-13: Vector force field of a single cell for particle in Figure 4-7. Core mass is located at (128.5, 128.5) which is to the left in this figure.

FIGURE 4–14
IMAGE FORCE vs. SOFTENING PARAMETER

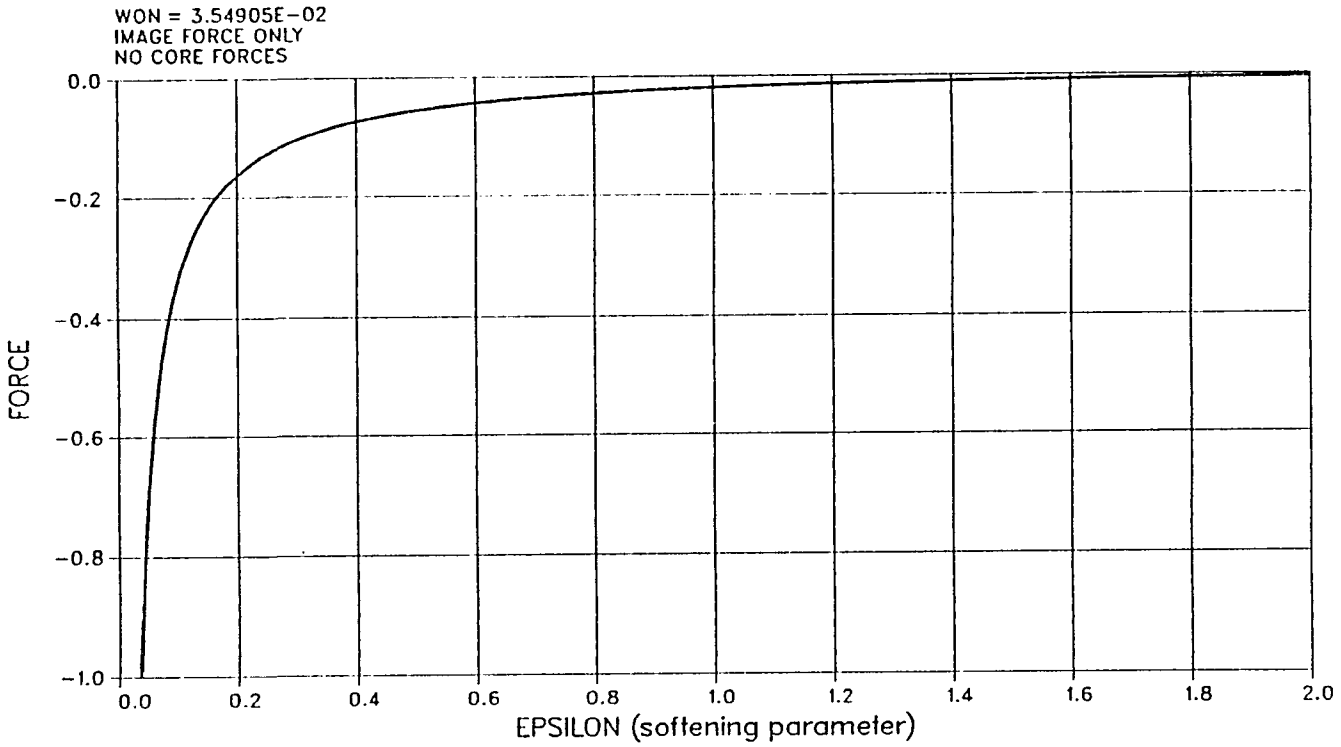

Fig 4-14: The magnitude of the force term q_ϵ in equations 4-33, 4-34, and 4-38 as a function of the softening parameter ϵ. The negative values show the force to be attractive.

FIGURE 4–15
ASQ = 30.00

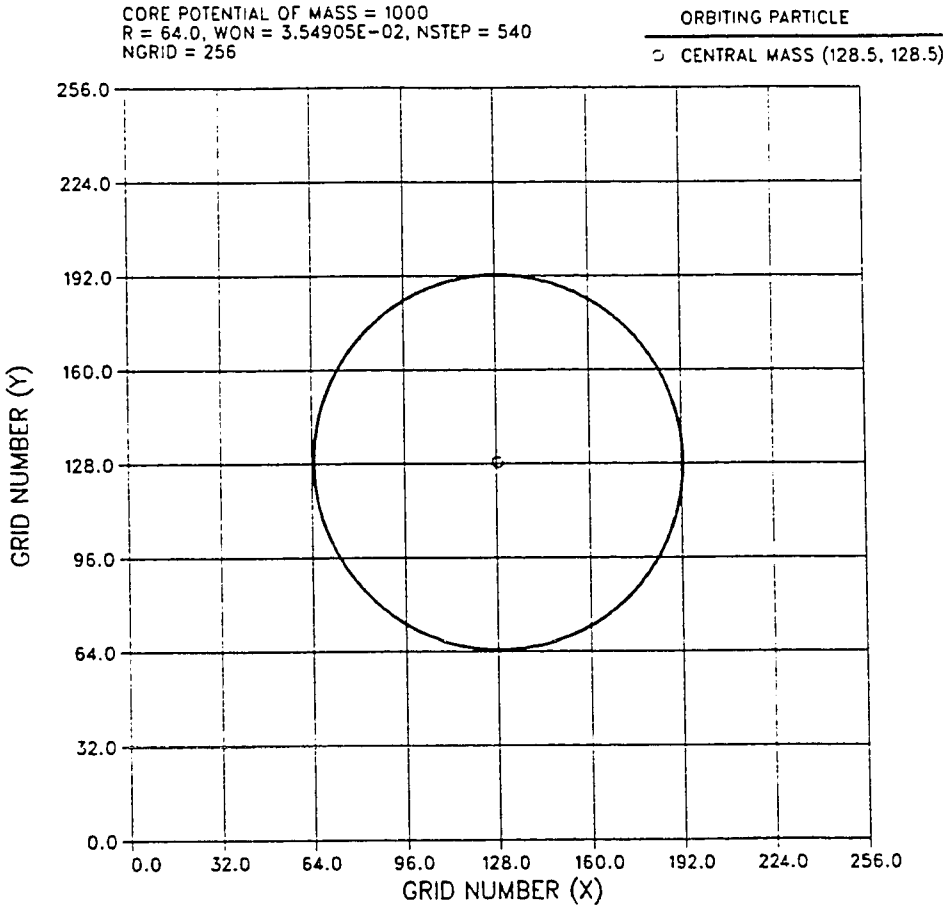

Fig 4-15: Particle in orbit about a massive core with a square of the softening parameter equal to 30.0 (ASQ = ϵ^2 = 30.0). Core mass is repressented by an effective potential equivalent to 1000 particles located at the center of the grid. Initial radius is 64, on a 256 X 256 grid. The orbital period is 540 time steps. Notice that the effect of the image force is not detectable.

FIGURE 4–16
PERCENT ERROR IN ORBITAL PATH (ASQ = 30)

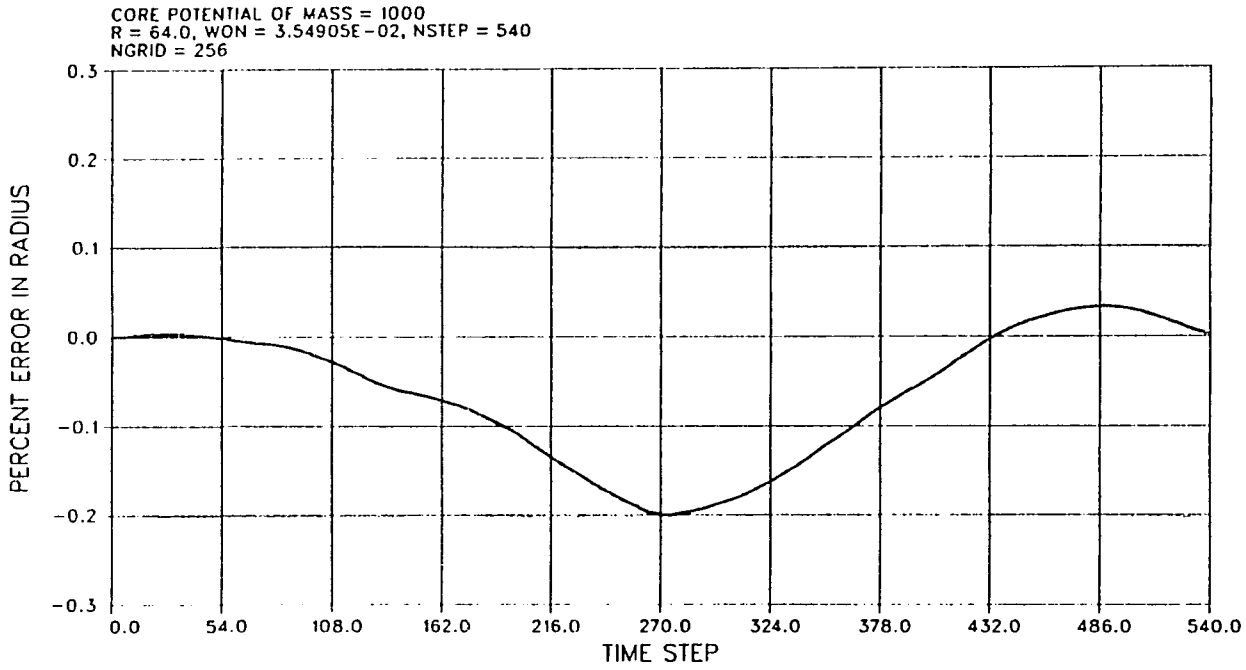

Fig 4-16: Percent error in orbital path for a particle in circular orbit with the square of the softening parameter equal to 30. This curve corresponds to the orbital path in figure 4-15.

FIGURE 4–17
SOFTENING OF ASQ = 30.0

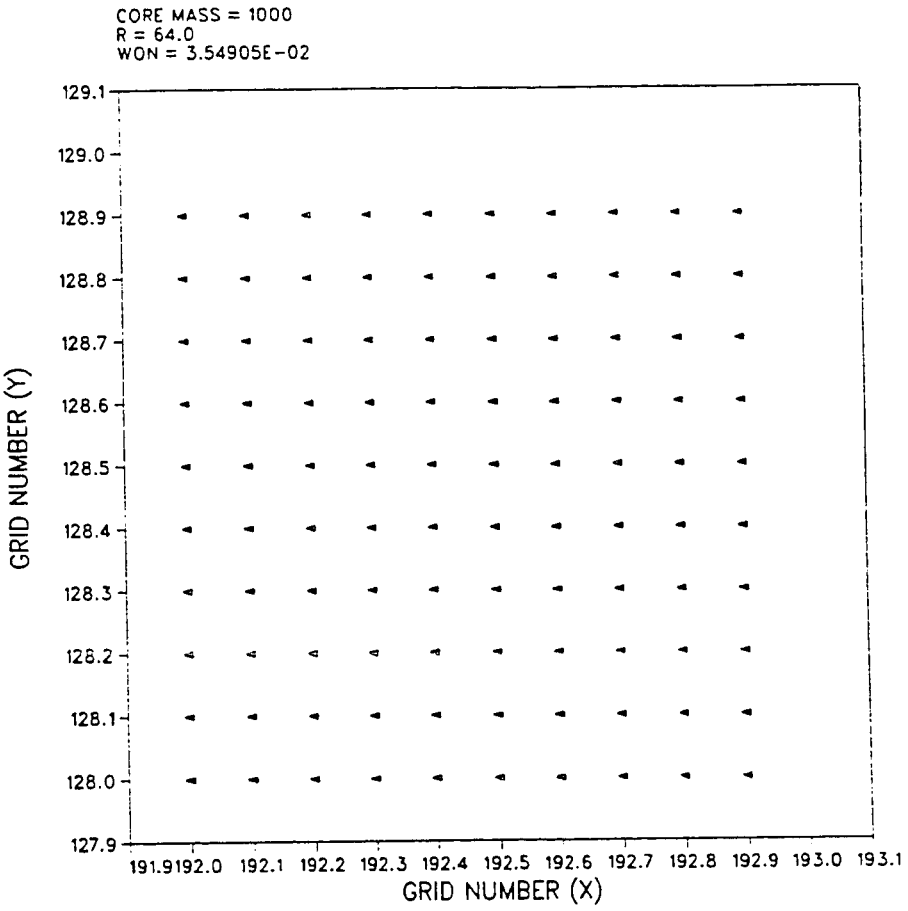

Fig 4-17: Vector force field of a single cell for particle in Figure 4-15. Core mass is located at (128.5, 128.5) which is to the left in this figure. Note that all the force vectors are in the direction of the central force. The effect of the image force has been supressed by using a large softening parameter.

FIGURE 4–18
ANGULAR MOMENTUM (PARTICLE WITH MASS REMOVED)

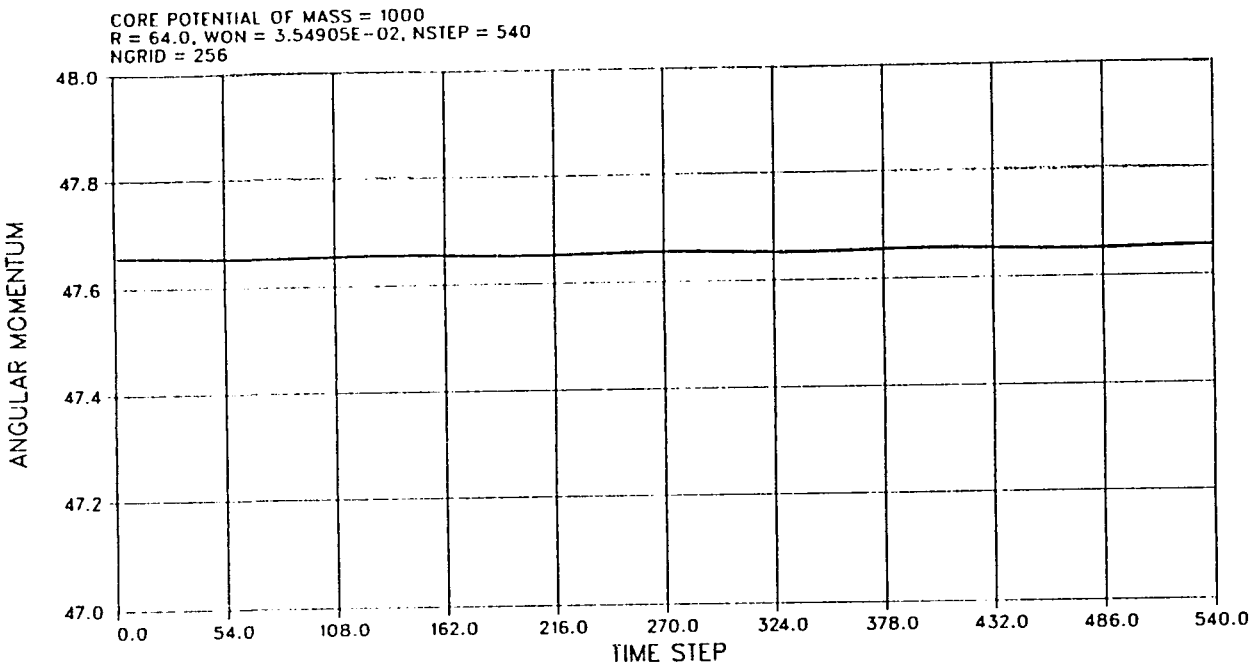

Fig 4-18: Angular momentum for the partiticle in figure 4-1. Conservation of angular momentum is represented as a constant value as a function of time.

FIGURE 4–19
ANGULAR MOMENTUM

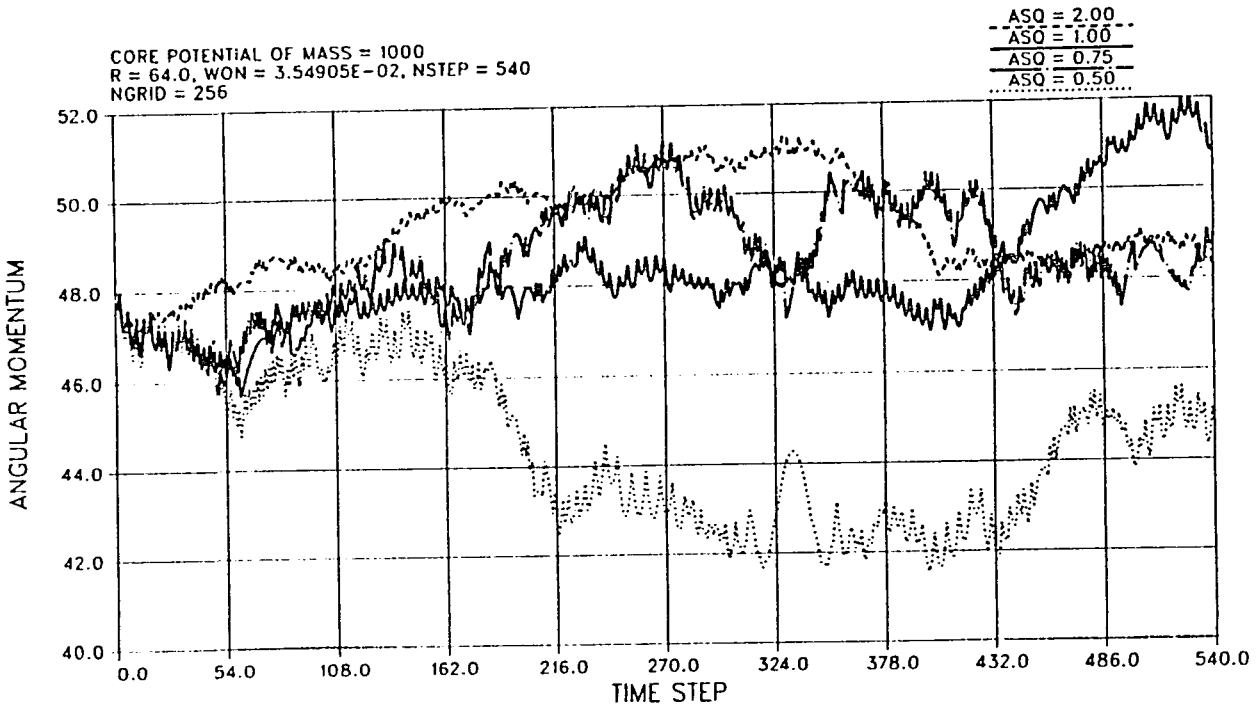

Fig 4-19: Angular momentum for the particles in figures 4-3, 4-4, 4-5, and 4-6 (ϵ^2 = 2.00, 1.00, 0.75, 0.50). Conservation of angular momentum is represented as a constant value as a function of time.

FIGURE 4–20
ANGULAR MOMENTUM (ASQ = 0.25)

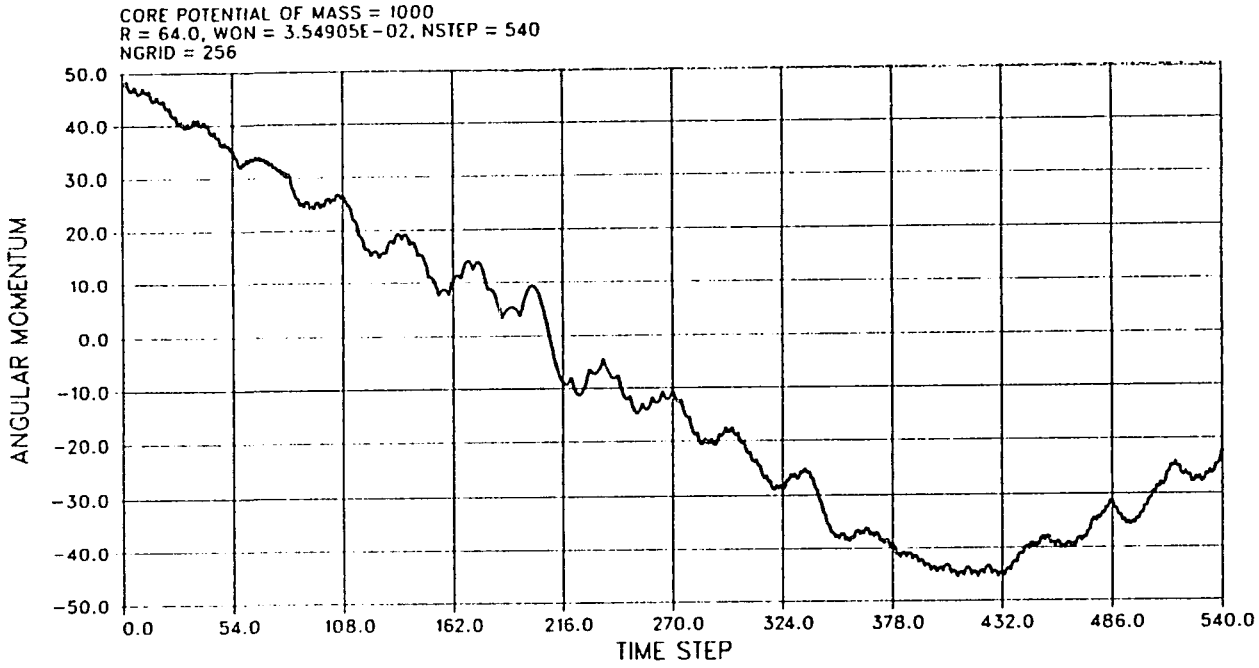

Fig 4-20: Angular momentum for the particle in figure 4-7 ($\epsilon^2 = 0.25$).

FIGURE 4–21
ANGULAR MOMENTUM (ASQ = 30.00)

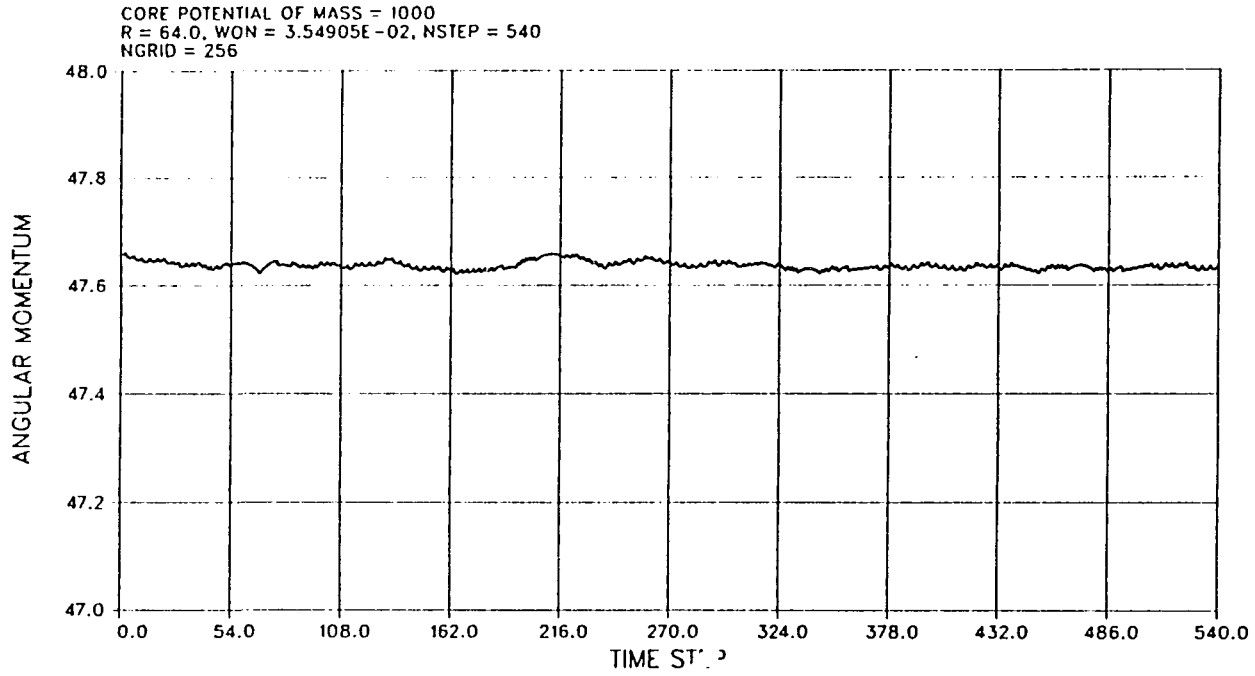

Fig 4-21: Angular momentum for the particle in figure 4-15 ($\epsilon^2 = 30.00$).

FIGURE 4–22
TOTAL ENERGY (PARTICLE WITH MASS REMOVED)

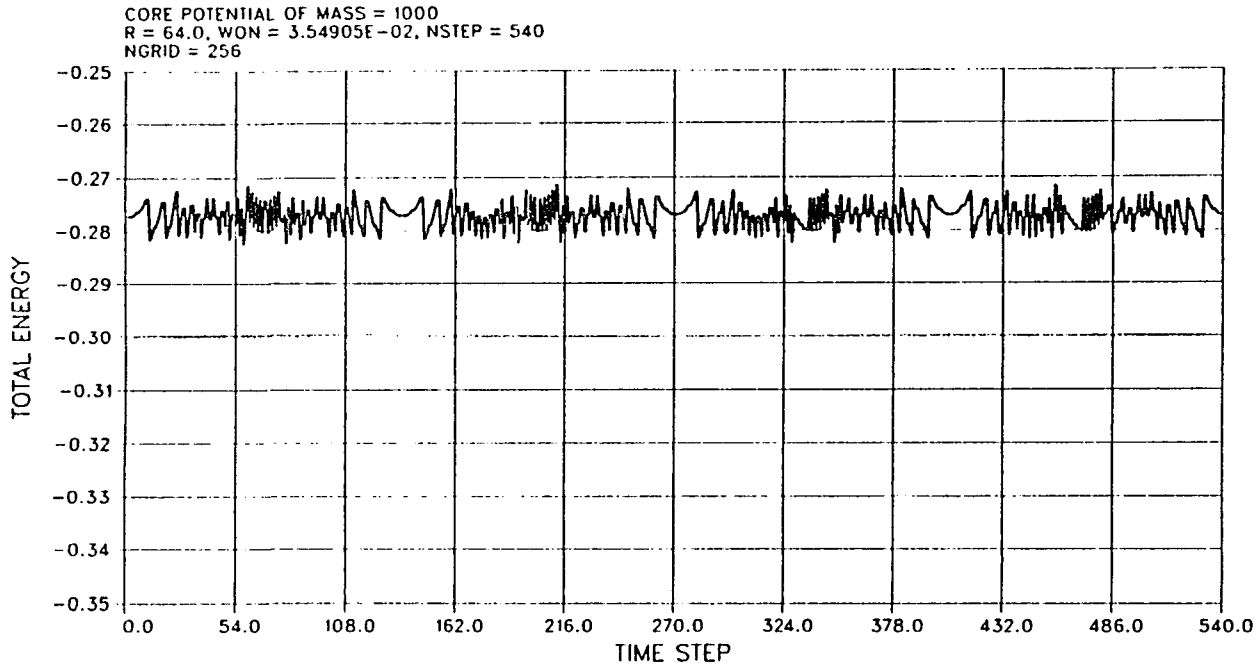

Fig 4-22: Total energy as a function of time for the particle in figure 4-1 were the mass of the particle was removed from the potenital calculation. A constant value indicates conservation of energy.

FIGURE 4-23
TOTAL ENERGY

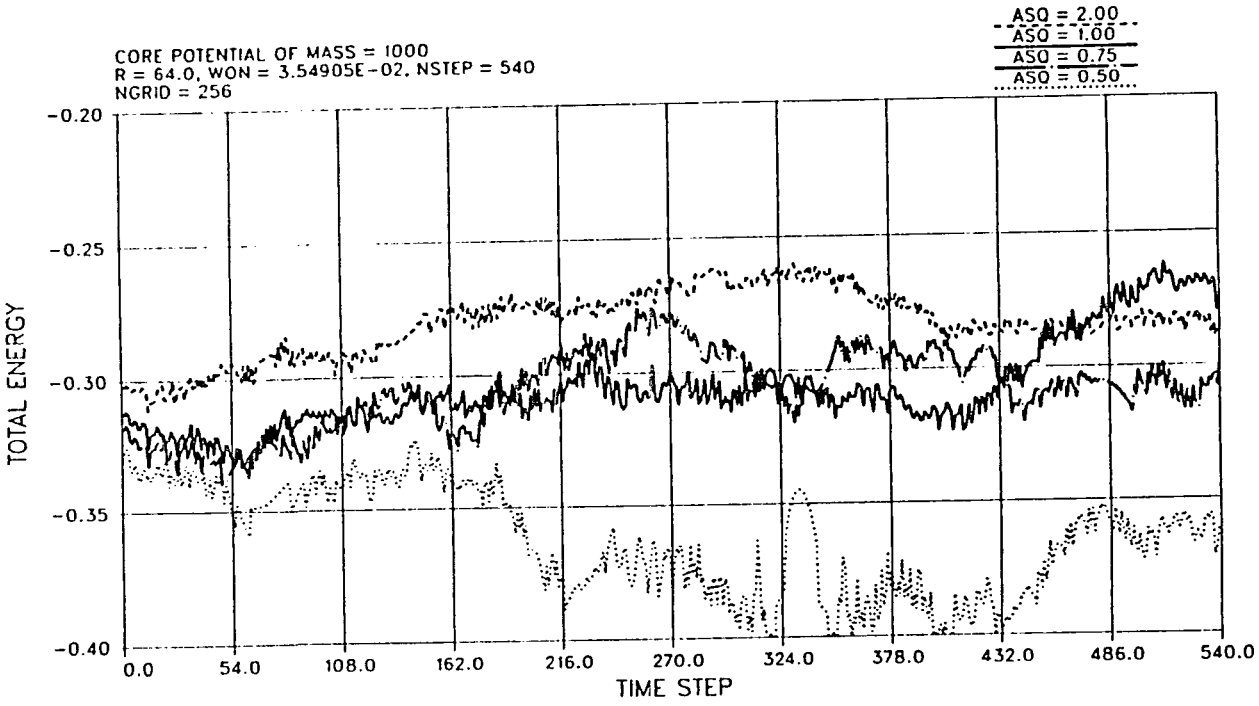

Fig 4-23: Total energy for the particles in figures 4-3, 4-4, 4-5, and 4-6 (ϵ^2 = 2.00, 1.00, 0.75, 0.50). Conservation of energy is represented as a constant value as a function of time.

FIGURE 4–24
TOTAL ENERGY (ASQ = 0.25)

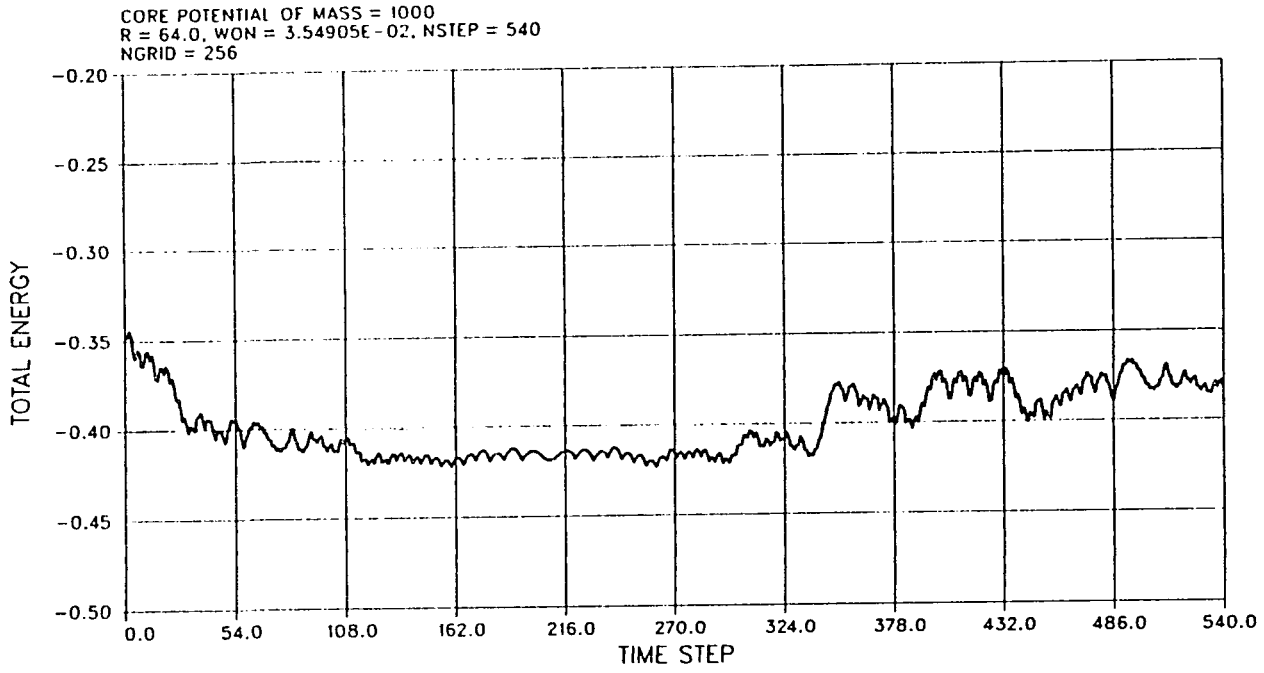

Fig 4-24: Total energy for the particle in figure 4-7 ($\epsilon^2 = 0.25$).

FIGURE 4–25
TOTAL ENERGY (ASQ = 30.00)

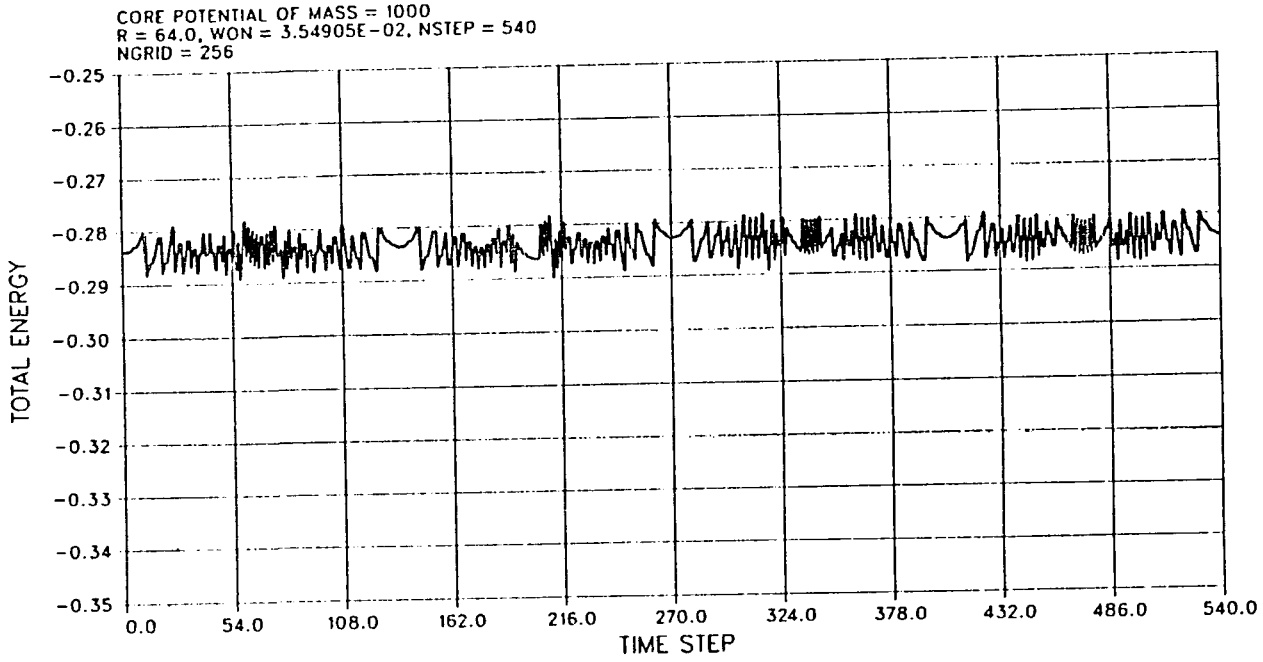

Fig 4-25: Total energy for the particle in figure 4-15 ($\epsilon^2 = 30.00$). Note the similarity to figure 4-22 were the mass of the particle was removed from the potential calculation.

FIGURE 5–1
10 BODY PROBLEM (ASQ = 5.00)

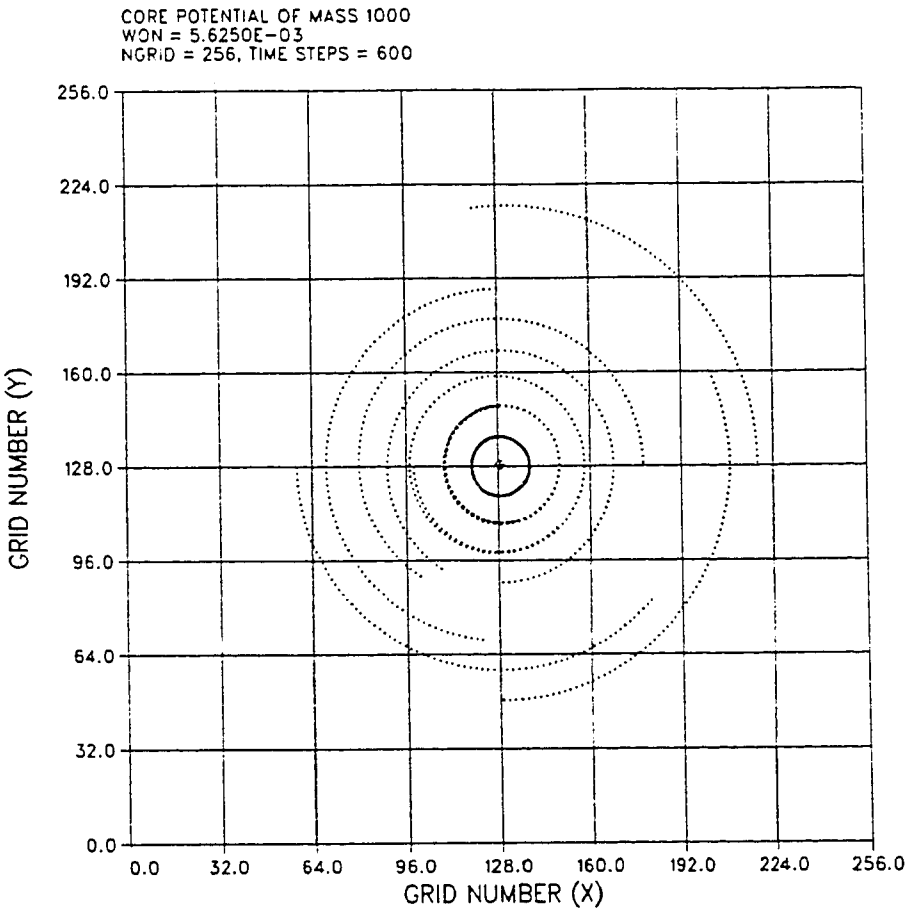

Fig 5-1: Orbital paths for nine particles in orbit about a core mass equivalent to 1000 particles. The total number of time steps is 600, for ASQ = ϵ^2 = 5.00, on a 256 X 256 grid.

FIGURE 5–2
10 BODY PROBLEM (ASQ = 0.25)

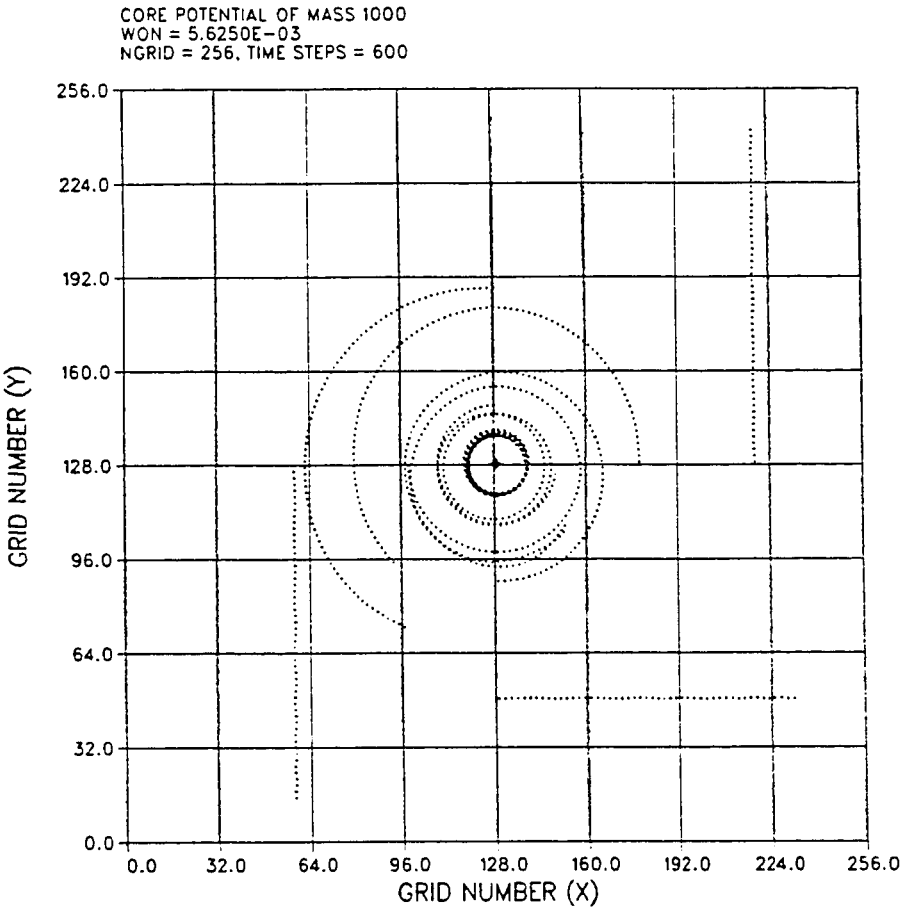

Fig 5-2: Orbital paths for nine particles in orbit about a core mass equivalent to 1000 particles. The total number of time steps is 600, for ASQ = ϵ^2 = 0.25, on a 256 X 256 grid.

FIGURE 5–3
ANGULAR MOMENTUM (10 body problem)

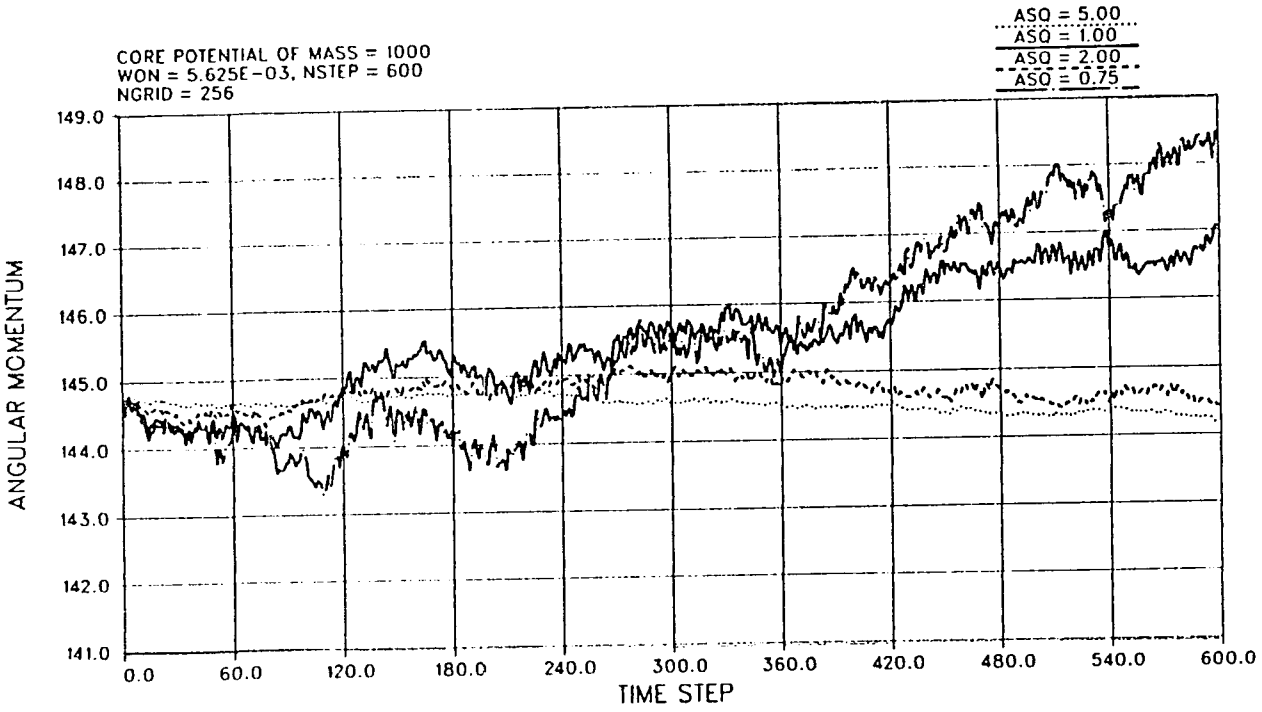

Fig 5-3: Angular momentum of the ten-body test for softening values of ASQ = ϵ^2 = 5.00, 2.00, 1.00, and 0.75. A constant value as a function of time representes conservation of angular momentum.

FIGURE 5–4
ANGULAR MOMENTUM (10 body problem)

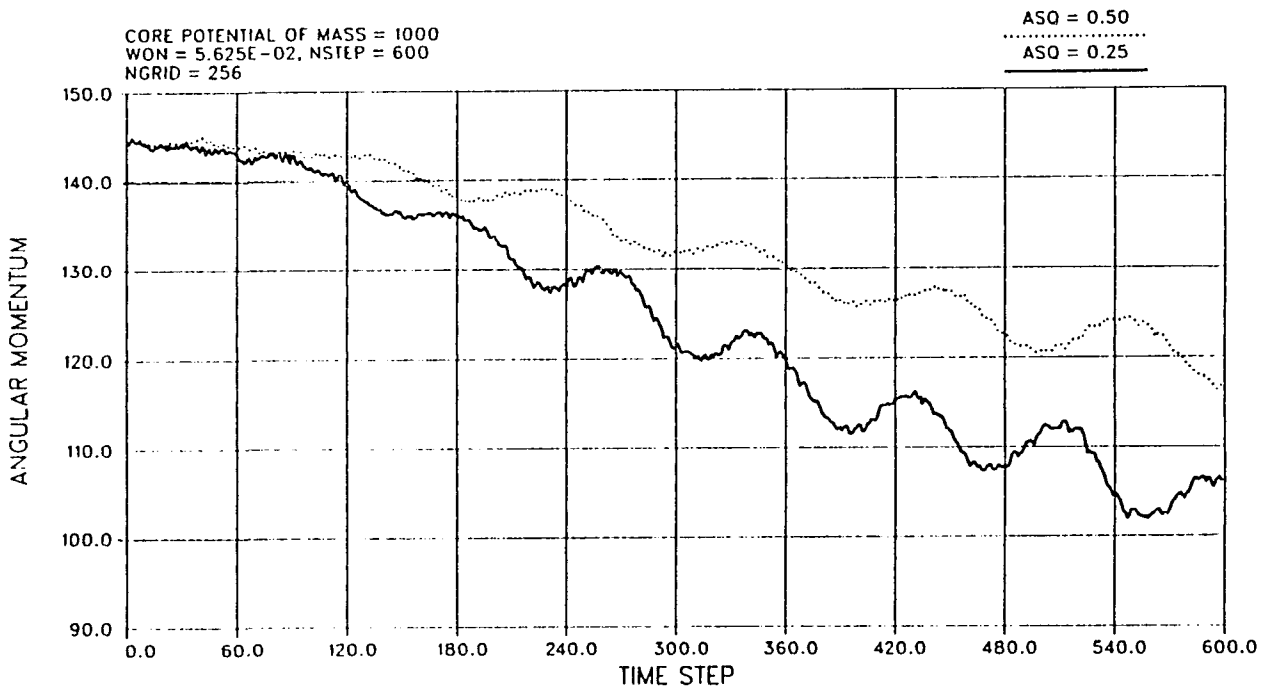

Fig 5-4: Angular momentum of the ten-body test for softening values of ASQ = ϵ^2 = 0.50, and 0.25. A constant value as a function of time representes conservation of angular momentum.

FIGURE 5–5
TOTAL ENERGY (10 body problem)

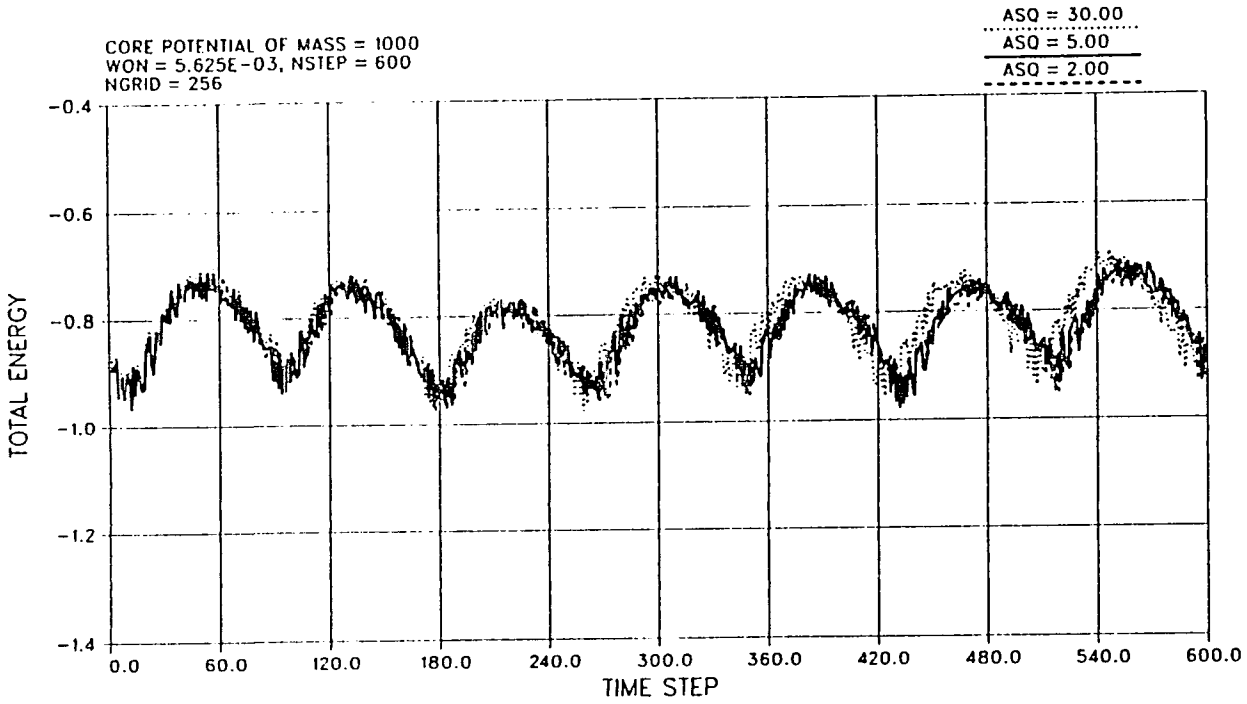

Fig 5-5: Total energy of the ten-body test for ASQ = ϵ^2 = 30.00, 5.00, and 2.00.

FIGURE 6-1

STABILITY DIAGRAM

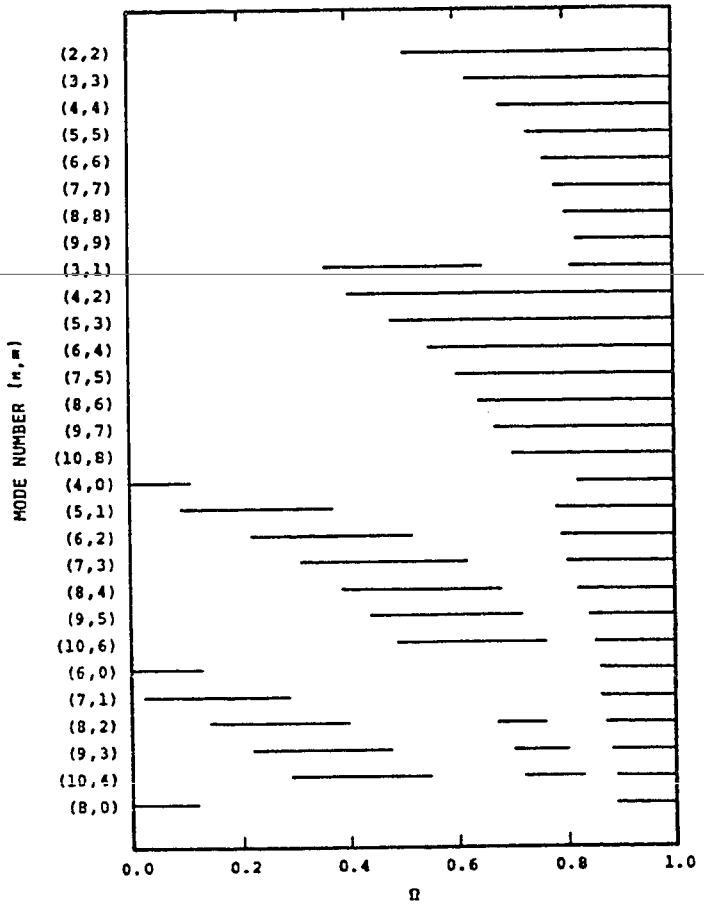

Fig 6-1: Stability diagram for the Kalnjas Disk Omega Models. Solid lines are regions of instability. From: A. J. Kalnajs, *Ap. J.* , 175, 70, 1972.

FIGURE 6-2a

HOHL, KALNAJS DISK PARTICLE PLOTS

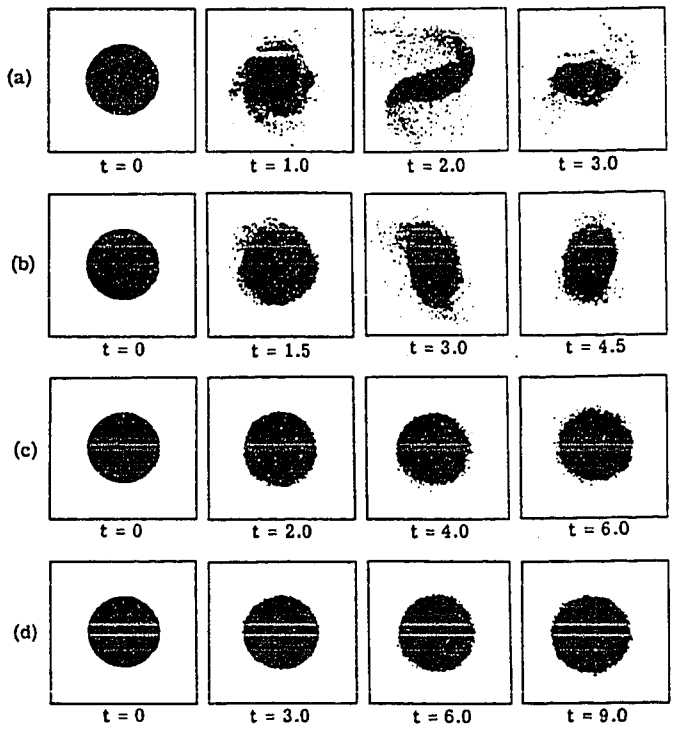

Fig 6-2a: Kalnajs disk particle plots where: (a) $\Omega = 0.8\Omega_0$, (b) $\Omega = 0.6\Omega_0$, (c) $\Omega = 0.4\Omega_0$, and (d) $\Omega = 0.0\Omega_0$. All times are referenced to the rotational period of the cold disk. From: F. Hohl, *J. Computational Physics*, **9**, 19, 1972.

FIGURE 6-2b

HOHL, KALNAJS DISK Q VALUES

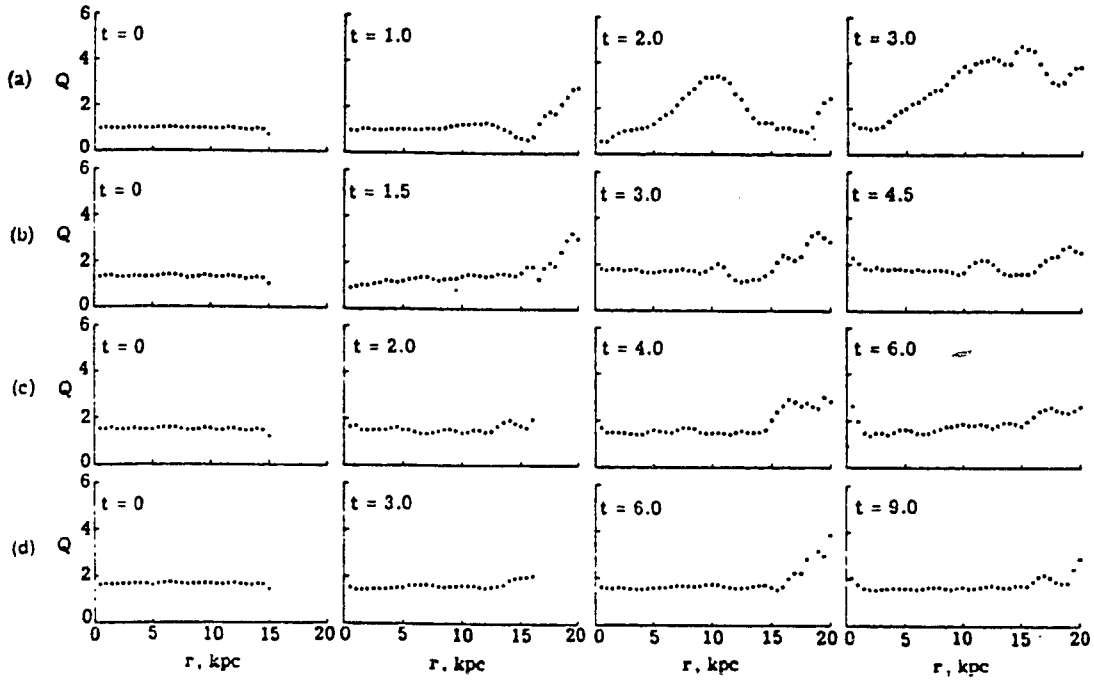

Fig 6-2b: Kalnajs disk Q values where: (a) $\Omega = 0.8\Omega_0$, (b) $\Omega = 0.6\Omega_0$, (c) $\Omega = 0.4\Omega_0$, and (d) $\Omega = 0.0\Omega_0$. All times are referenced to the rotational period of the cold disk. From: F. Hohl, *J. Computational Physics*, 9, 23, 1972.

FIGURE 6-2c

HOHL, KALNAJS DISK VELOCITY DISPERSION

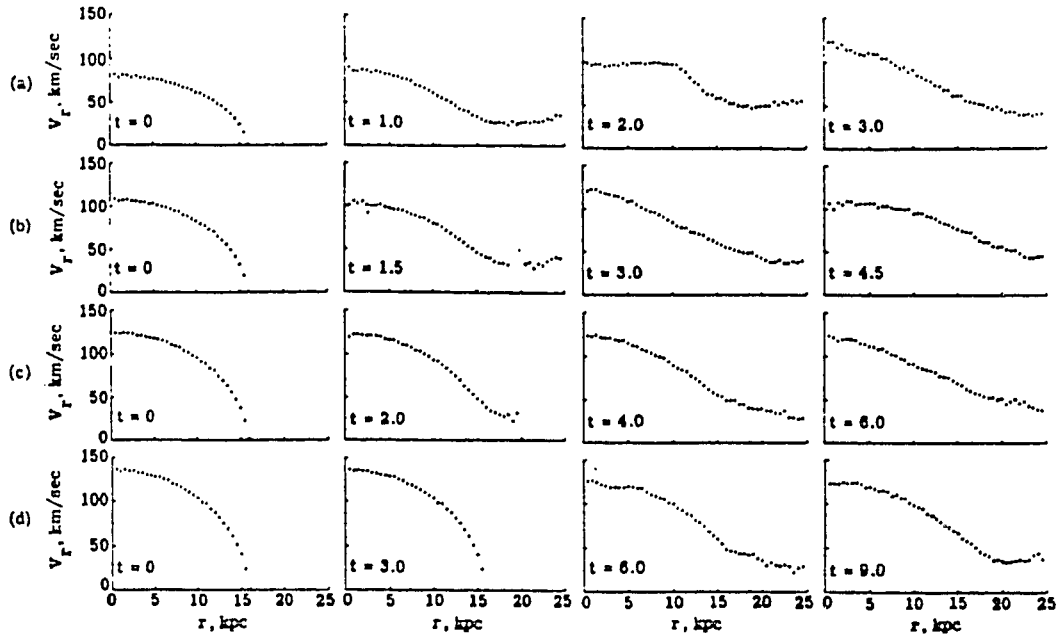

Fig 6-2c: Kalnajs disk radial velocity dispersions where: (a) $\Omega = 0.8\Omega_0$, (b) $\Omega = 0.6\Omega_0$, (c) $\Omega = 0.4\Omega_0$, and (d) $\Omega = 0.0\Omega_0$. All times are referenced to the rotational period of the cold disk.
From: F. Hohl, *J. Computational Physics*, 9, 22, 1972.

FIGURE 6-2d

HOHL, KALNAJS DISK SURFACE DENSITY

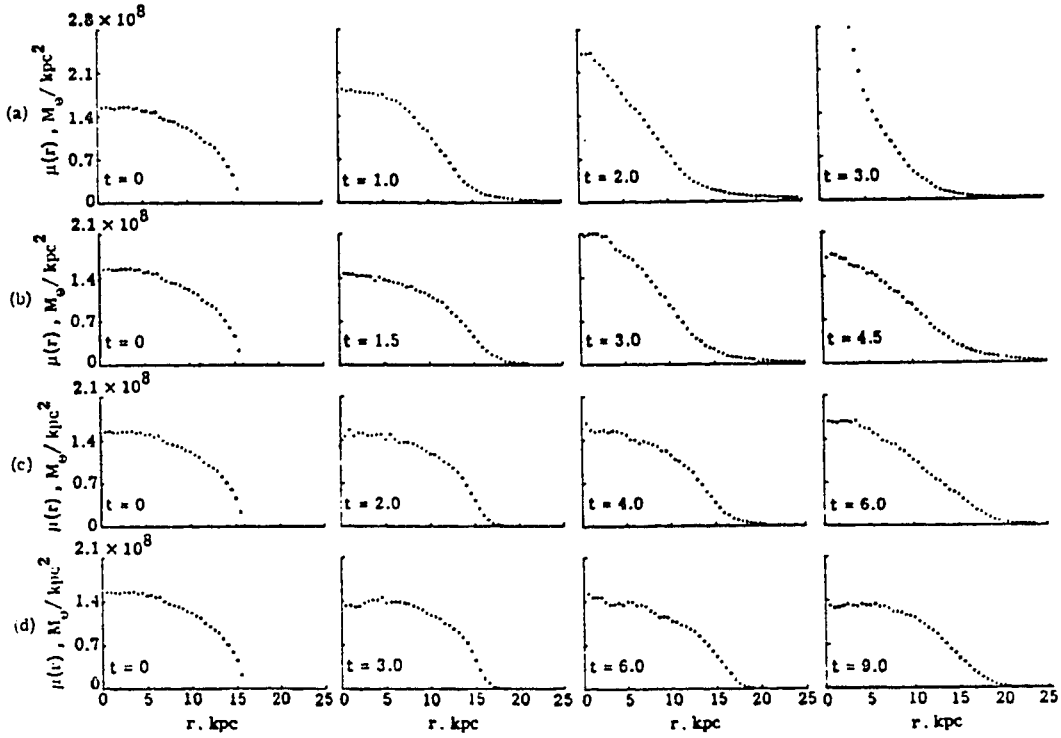

Fig 6-2d: Kalnajs disk surface density where: (a) $\Omega = 0.8\Omega_0$, (b) $\Omega = 0.6\Omega_0$, (c) $\Omega = 0.4\Omega_0$, and (d) $\Omega = 0.0\Omega_0$. All times are referenced to the rotational period of the cold disk. From: F. Hohl, *J. Computational Physics*, 9, 22, 1972.

FIGURE 6-3

POLAR GRID SOLUTION: $\Omega = 0.8\Omega_0$

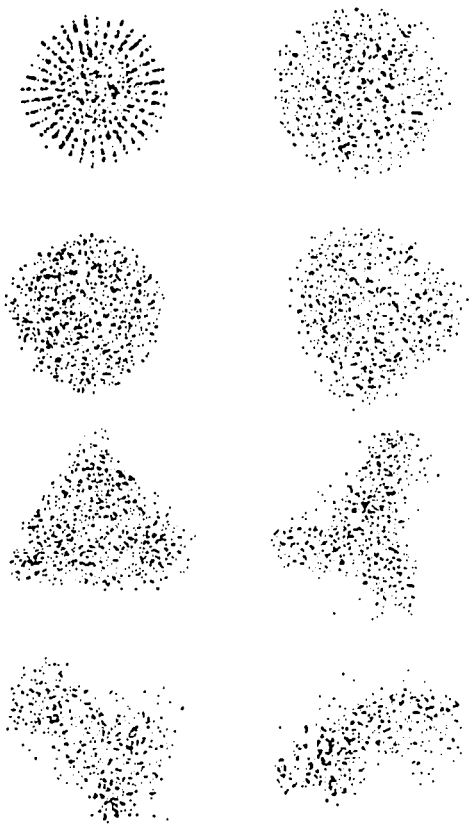

Fig 6-3: Polar grid solution to a Kalnajs disk for $\Omega = 0.8\Omega_0$. From top left to bottom right, the periods are 0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, and 3.5 equivalent cold disk rotations. Period 3.0 and 3.5 show the barlike structure. From: R. H. Miller, *J. Computational Physics*, 21, 421, 1976.

FIGURE 6–4
SURFACE DENSITY: INITIAL LOAD: OMEGA = 0.4

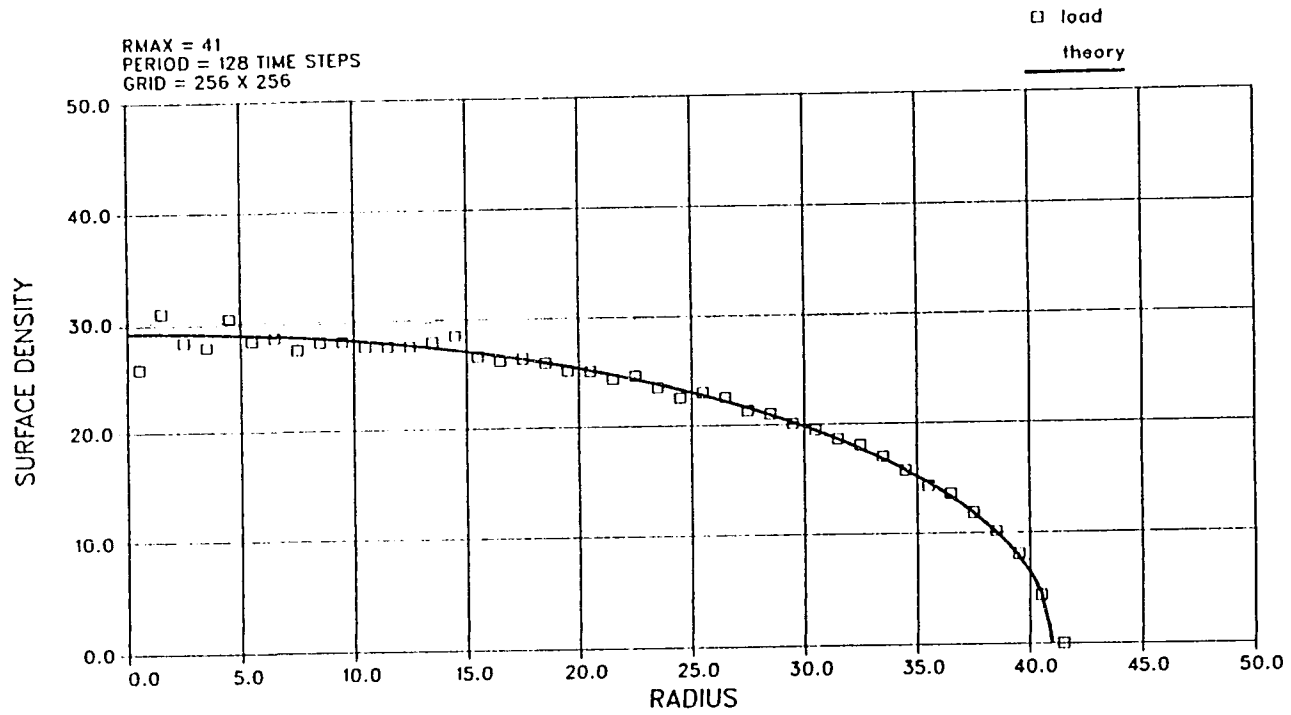

Fig 6-4: Theoretical surface density of a Kalnajs disk is shown as the solid curve. Squares show the surface density calculated from the random load, sorted in rings one grid unit wide.

FIGURE 6–5
RADIAL VELOCITY DISPERSION: INITAL LOAD: OMEGA = 0.4

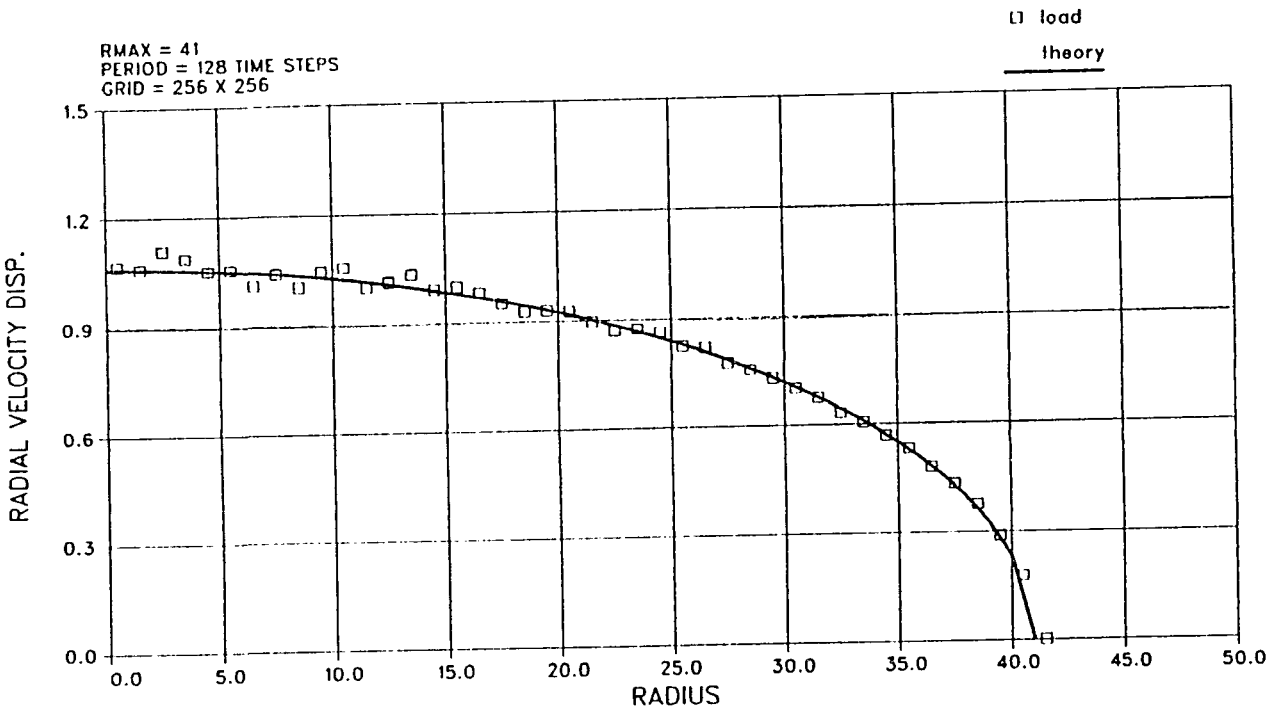

Fig 6-5: Theoretical radial velocity dispersion of a Kalnajs disk, $\Omega = 0.4\Omega_0$ is shown as the solid curve. Squares show the radial velocity dispersion calculated from the random load, sorted in rings one grid unit wide.

FIGURE 6–6
RADIAL VELOCITY DISPERSION: INITAL LOAD: OMEGA = 0.8

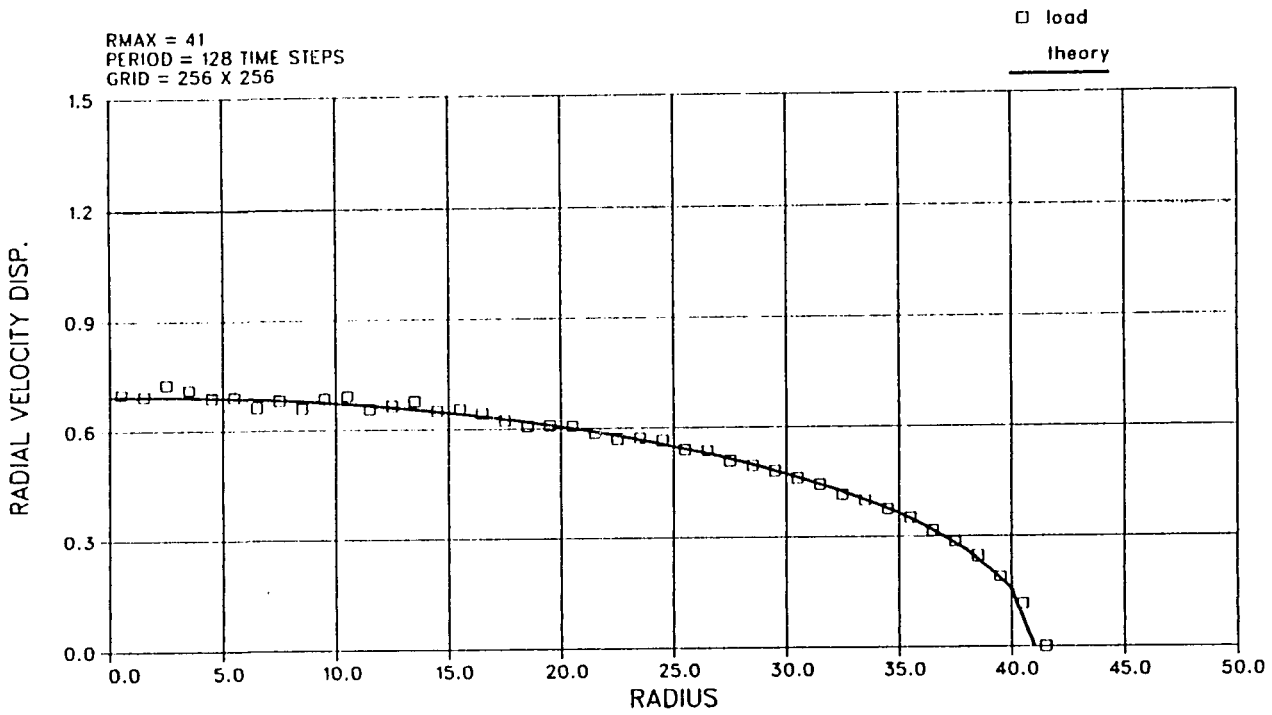

Fig 6-6: Theoretical radial velocity dispersion of a Kalnajs disk, $\Omega = 0.8\Omega_0$ is shown as the solid curve. Squares show the radial velocity dispersion calculated from the random load, sorted in rings one grid unit wide.

FIGURE 6–7
PERIOD = 0: OMEGA = 0.4

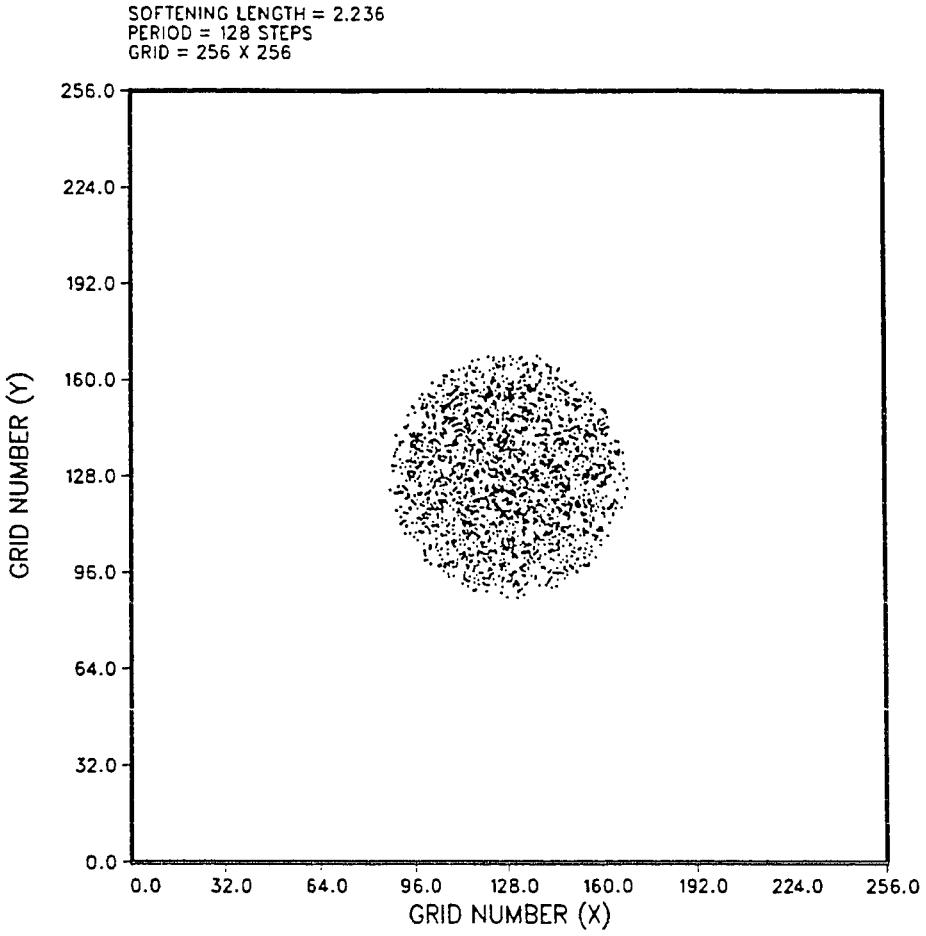

Fig 6-7: Particle plot for the $\Omega = 0.4\Omega_0$ Kalnajs disk omega model. This figure shows the initial load.

FIGURE 6-8
PERIOD = 1: OMEGA = 0.4

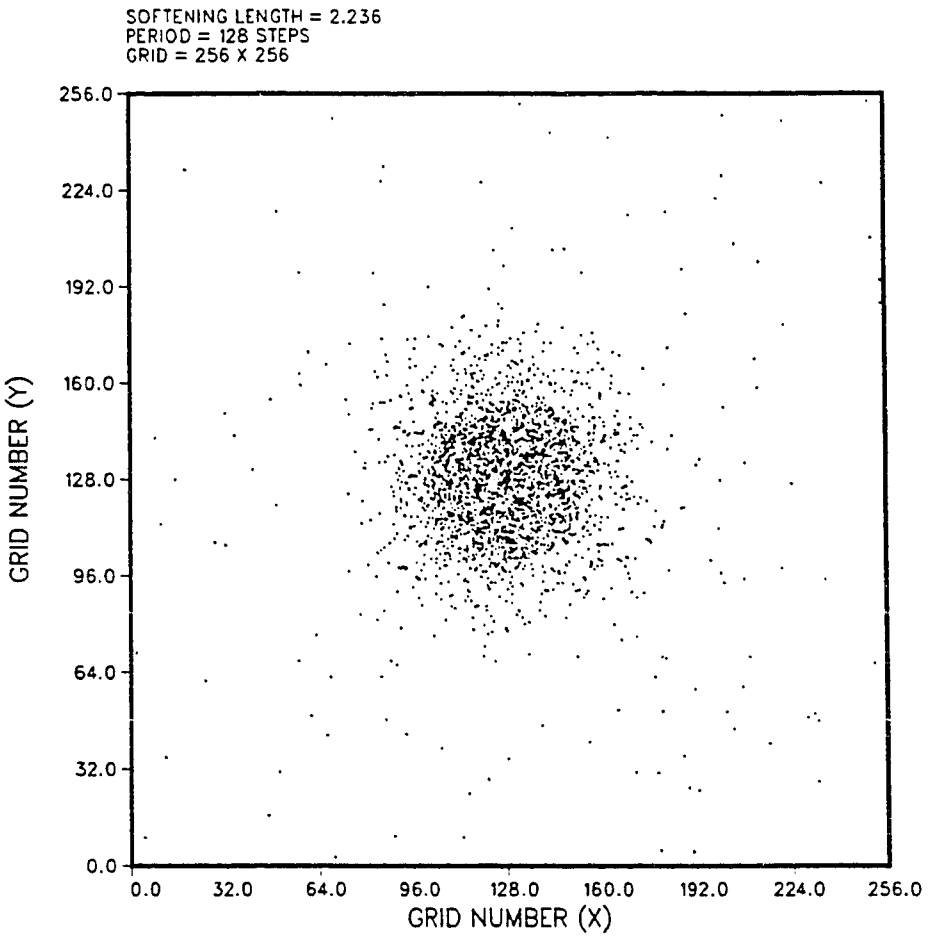

Fig 6-8: Particle plot for the $\Omega = 0.4\Omega_0$ Kalnajs disk omega model. This figure shows the disk after one period of revolution. One period is 128.8 time steps. The value of softening is $\epsilon^2 = 5$. The computational grid is 256 X 256, and 1/40th of the particles used are plotted.

FIGURE 6–9
PERIOD = 2: OMEGA = 0.4

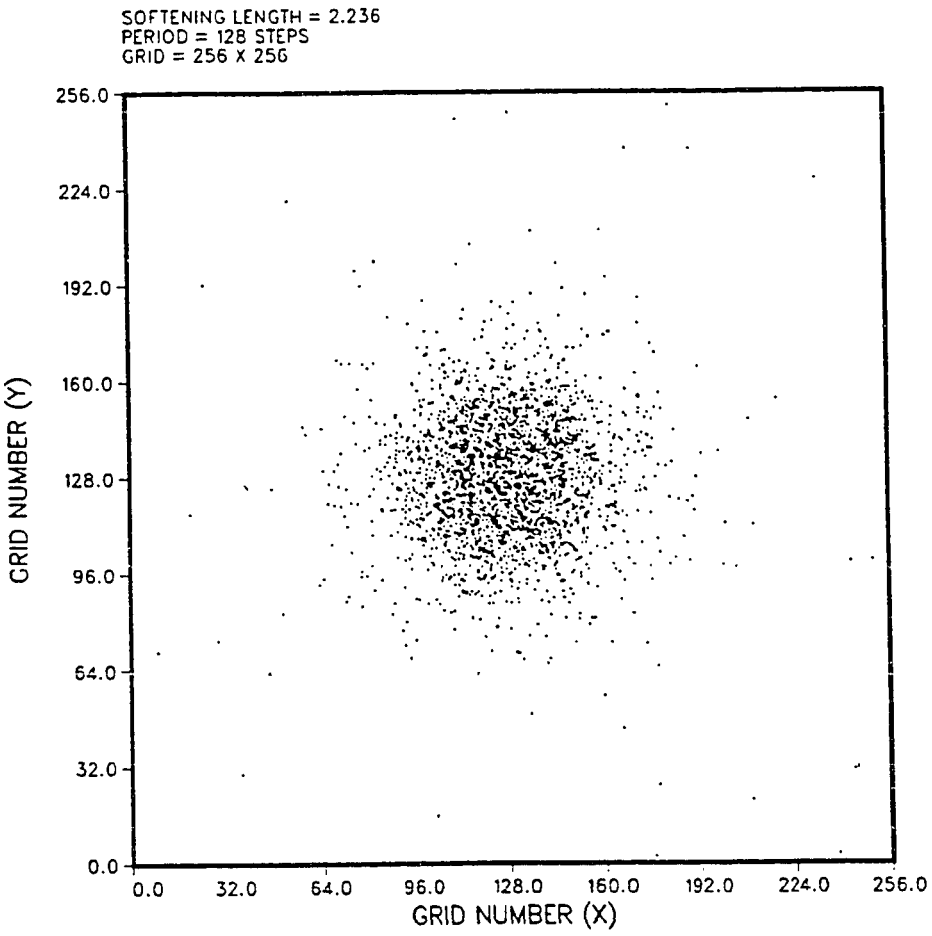

Fig 6-9: Particle plot for the $\Omega = 0.4\Omega_0$ Kalnajs disk omega model. This figure shows the disk after two periods of revolution. One period is 128.8 time steps. The value of softening is $\epsilon^2 = 5$. The computational grid is 256 X 256, and 1/40th of the particles used are plotted.

FIGURE 6–10
PERIOD = 3: OMEGA = 0.4

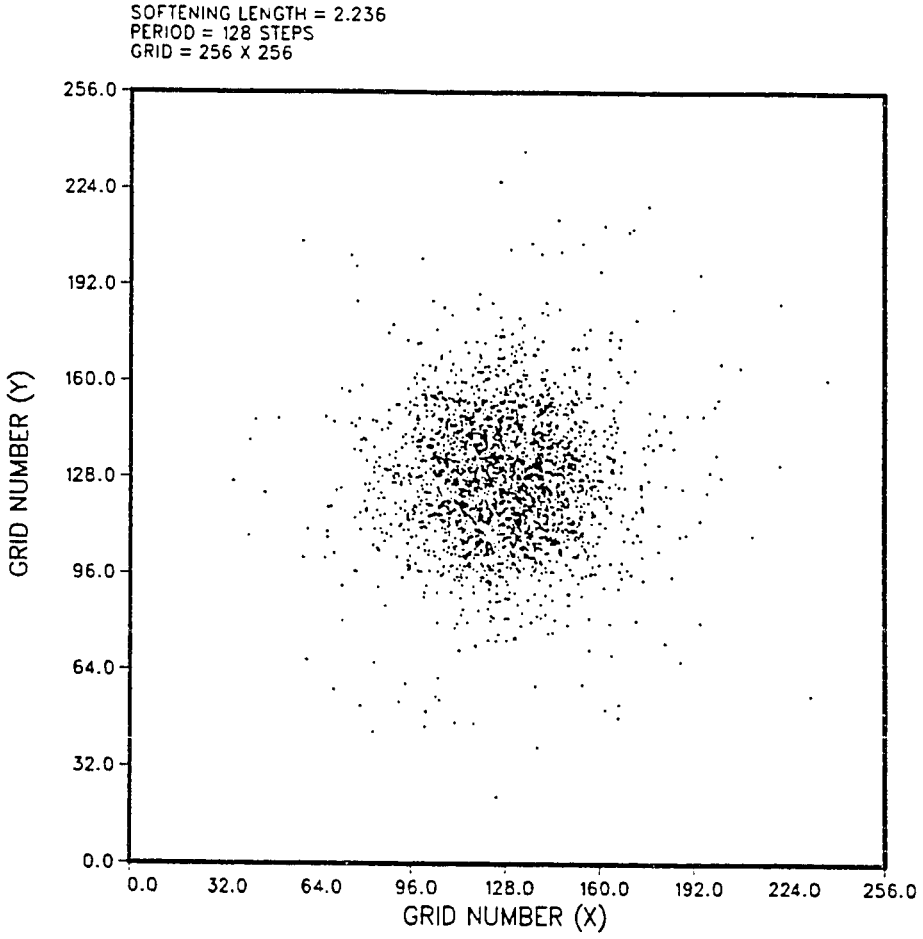

Fig 6-10: Particle plot for the $\Omega = 0.4\Omega_0$ Kalnajs disk omega model. This figure shows the disk after three periods of revolution. One period is 128.8 time steps. The value of softening is $\epsilon^2 = 5$. The computational grid is 256 X 256, and 1/40th of the particles used are plotted.

FIGURE 6–11
PERIOD = 4: OMEGA = 0.4

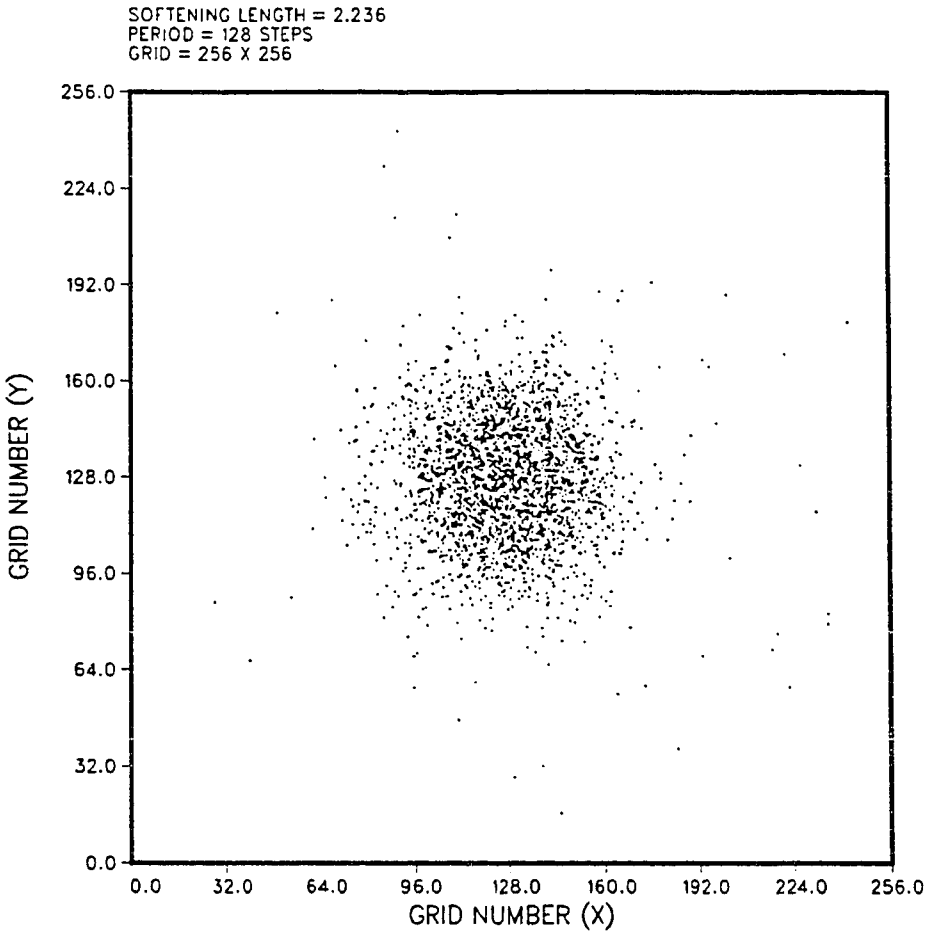

Fig 6-11: Particle plot for the $\Omega = 0.4\Omega_0$ Kalnajs disk omega model. This figure shows the disk after four periods of revolution. One period is 128.8 time steps. The value of softening is $\epsilon^2 = 5$. The computational grid is 256 X 256, and 1/40th of the particles used are plotted.

FIGURE 6–12
PERIOD = 5: OMEGA = 0.4

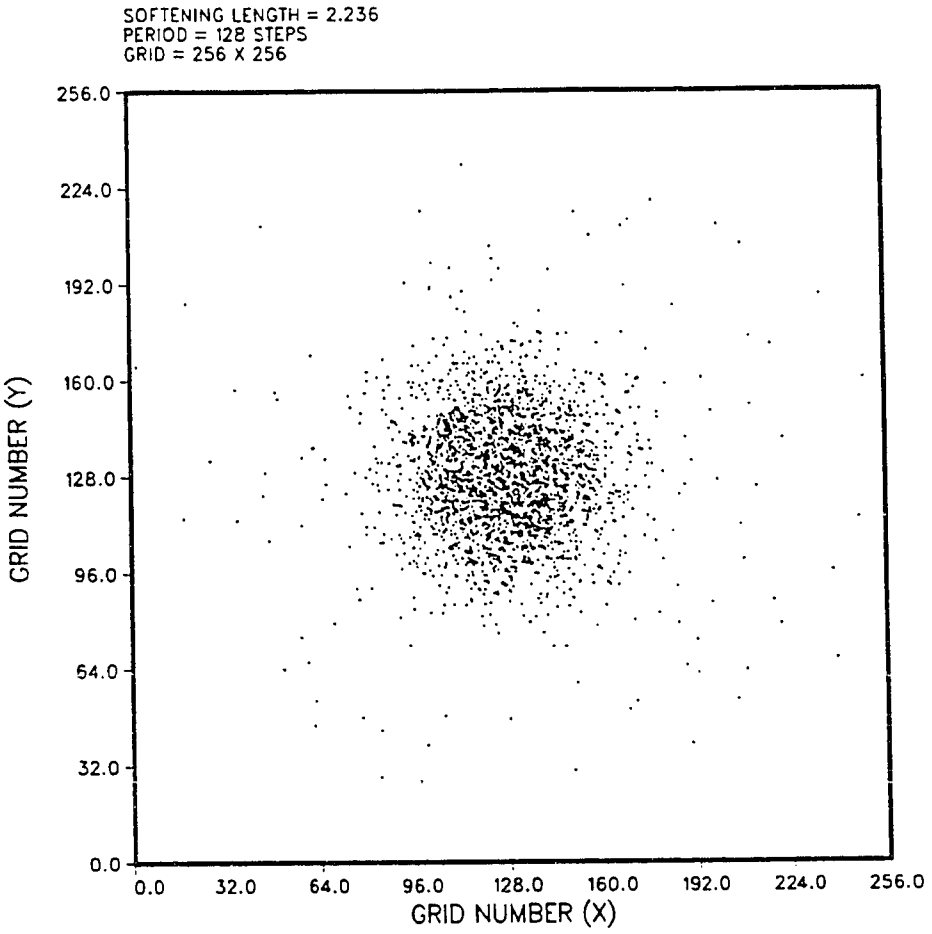

Fig 6-12: Particle plot for the $\Omega = 0.4\Omega_0$ Kalnajs disk omega model. This figure shows the disk after five periods of revolution. One period is 128.8 time steps. The value of softening is $\epsilon^2 = 5$. The computational grid is 256 X 256, and 1/40th of the particles used are plotted.

FIGURE 6–13
PERIOD = 6: OMEGA = 0.4

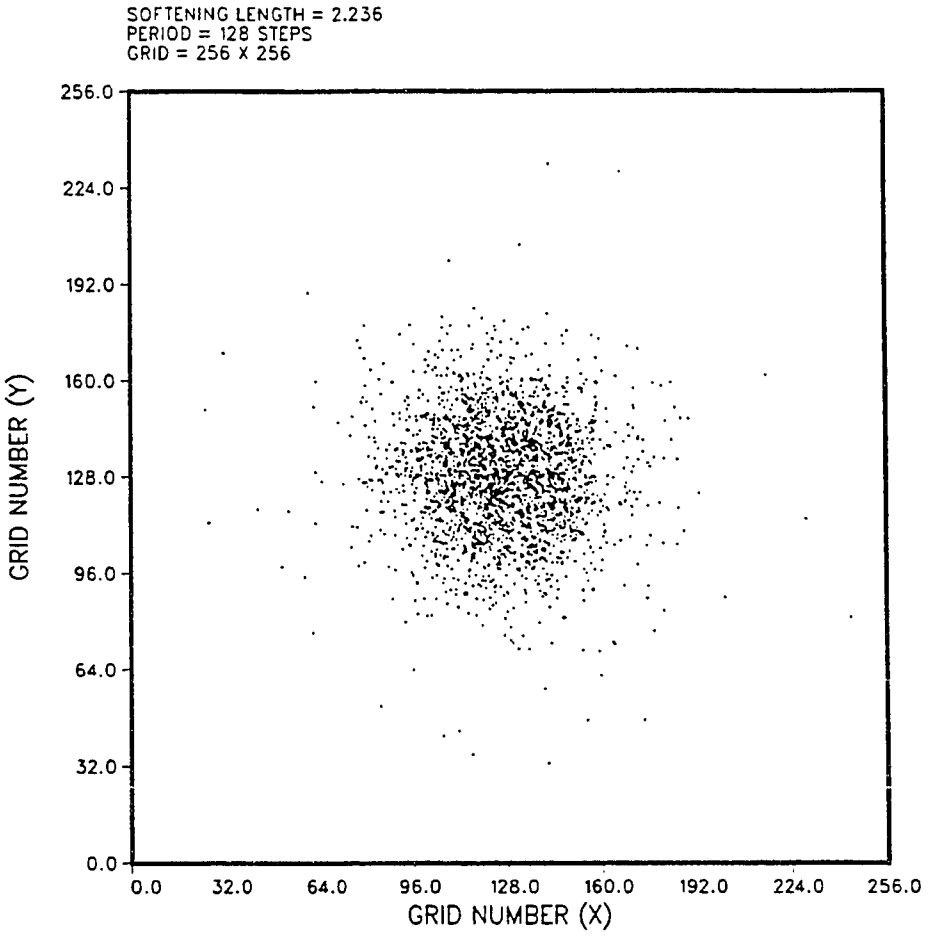

Fig 6-13: Particle plot for the $\Omega = 0.4\Omega_0$ Kalnajs disk omega model. This figure shows the disk after six periods of revolution. One period is 128.8 time steps. The value of softening is $\epsilon^2 = 5$. The computational grid is 256 X 256, and 1/40th of the particles used are plotted.

FIGURE 6–14
VELOCITY DISPERSION: PERIOD = 0: OMEGA = 0.4

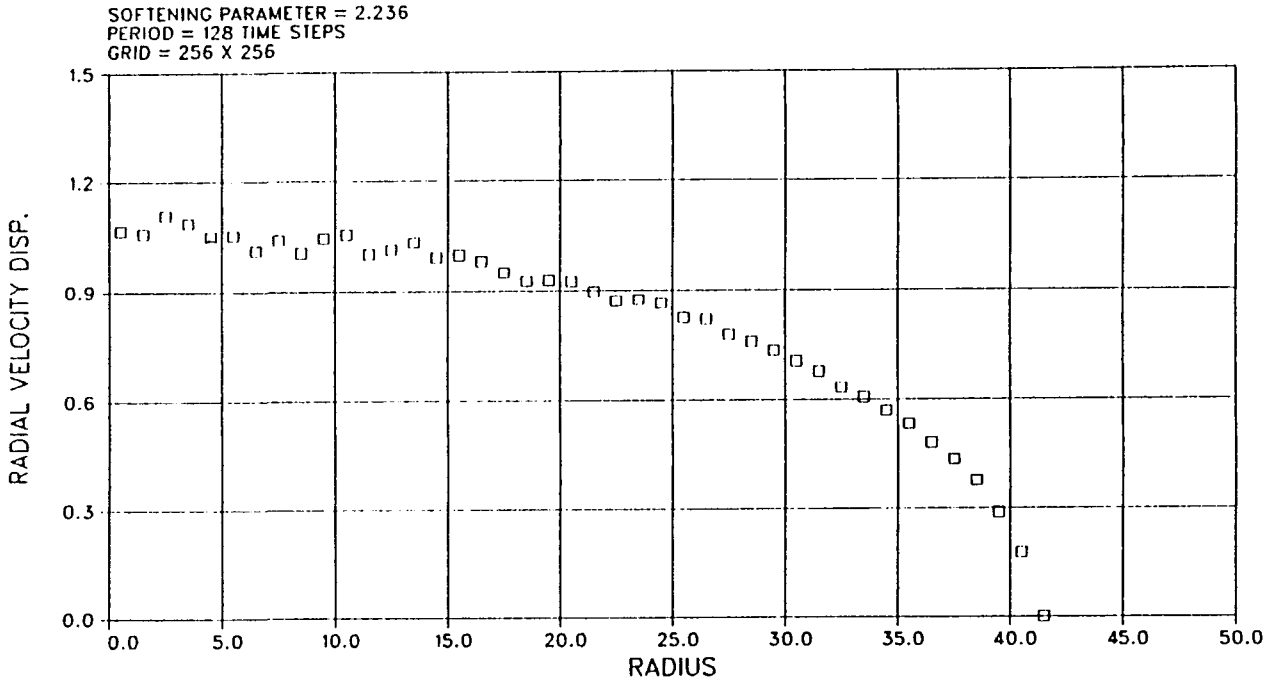

Fig 6-14: Radial velocity dispersion for the $\Omega = 0.4\Omega_0$ Kalnajs disk as calculated by CART2D for period = 0. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–15
VELOCITY DISPERSION: PERIOD = 1: OMEGA = 0.4

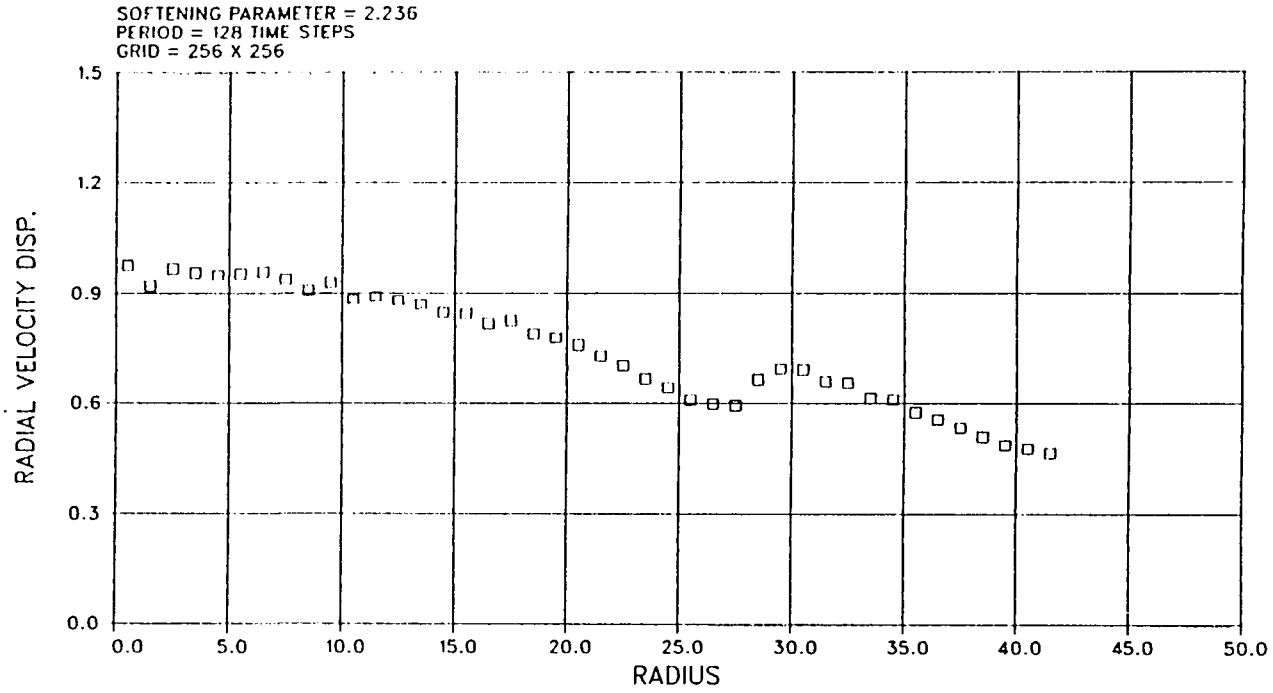

Fig 6-15: Radial velocity dispersion for the $\Omega = 0.4\Omega_0$ Kalnajs disk as calculated by CART2D for period = 1. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–16
VELOCITY DISPERSION: PERIOD = 2: OMEGA = 0.4

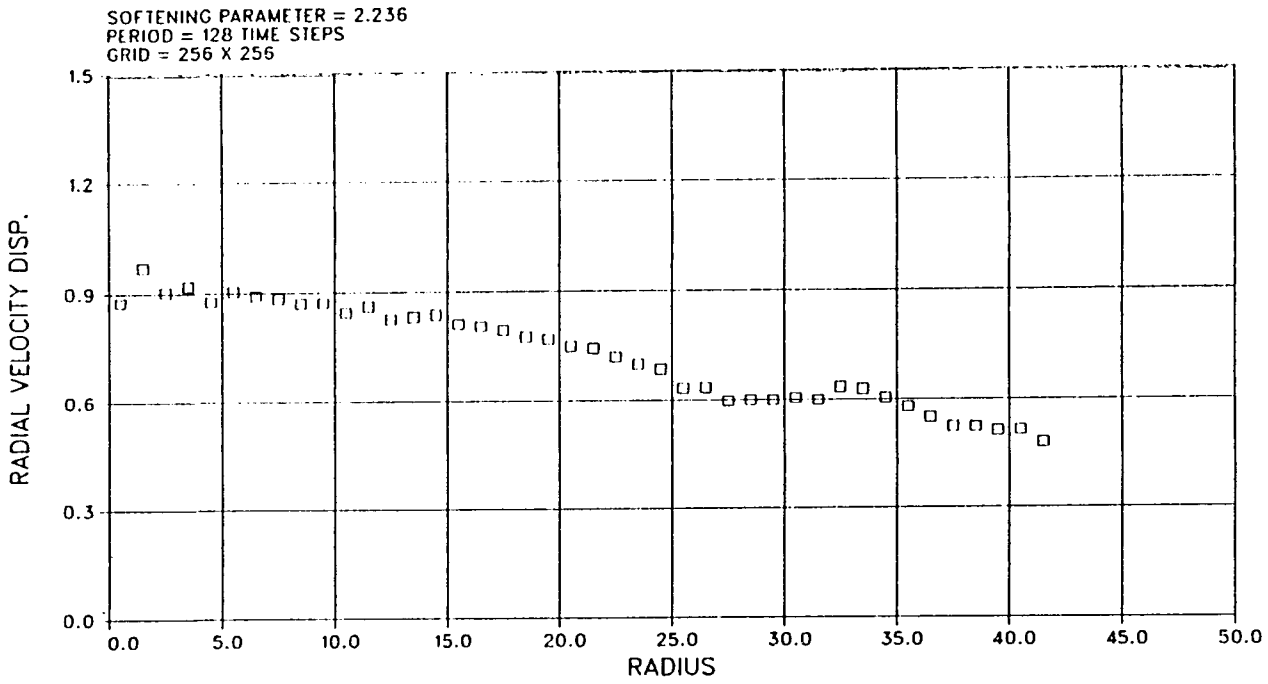

Fig 6-16: Radial velocity dispersion for the $\Omega = 0.4\Omega_0$ Kalnajs disk as calculated by CART2D for period = 2. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–17
VELOCITY DISPERSION: PERIOD = 3: OMEGA = 0.4

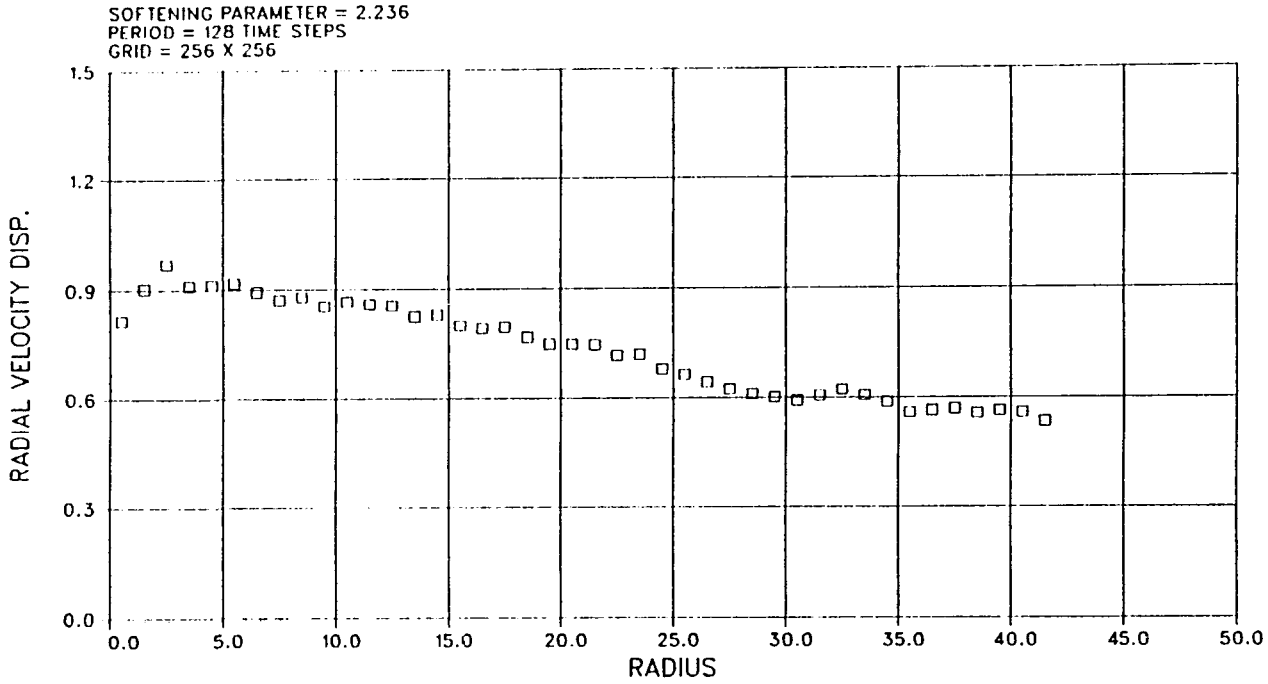

Fig 6-17: Radial velocity dispersion for the $\Omega = 0.4\Omega_0$ Kalnajs disk as calculated by CART2D for period = 3. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–18
VELOCITY DISPERSION: PERIOD = 4: OMEGA = 0.4

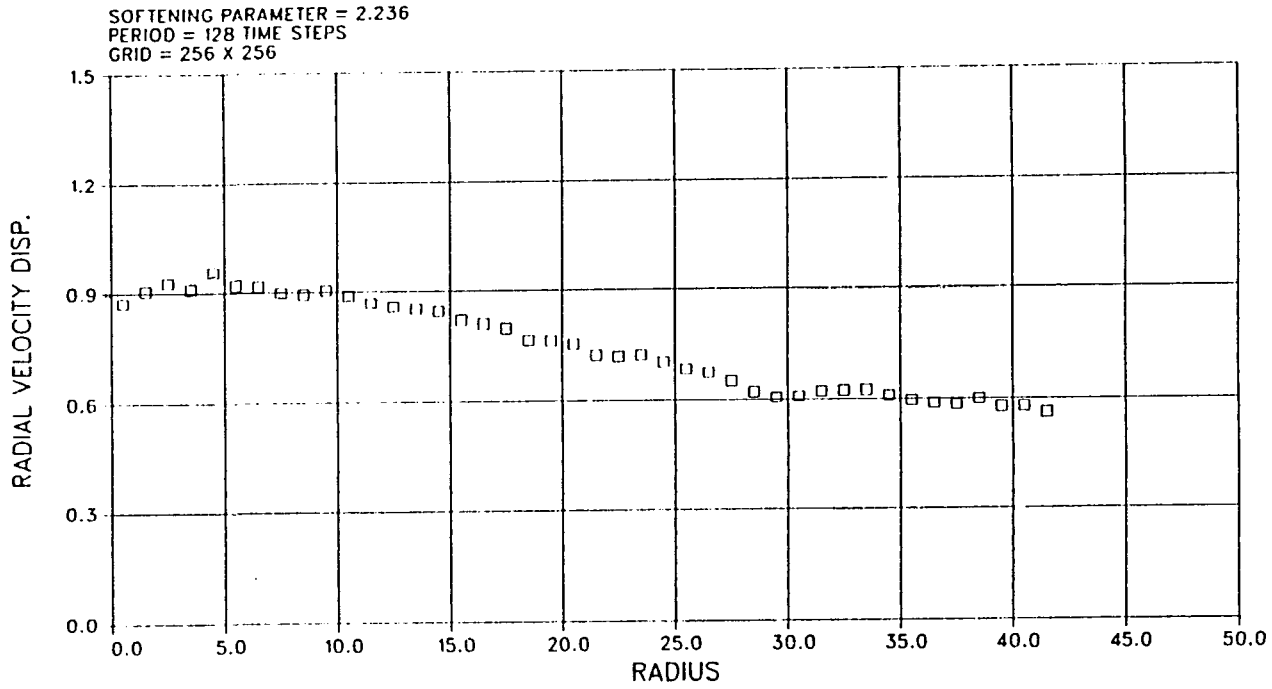

Fig 6-18: Radial velocity dispersion for the $\Omega = 0.4\Omega_0$ Kalnajs disk as calculated by CART2D for period = 4. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–19
VELOCITY DISPERSION: PERIOD = 5: OMEGA = 0.4

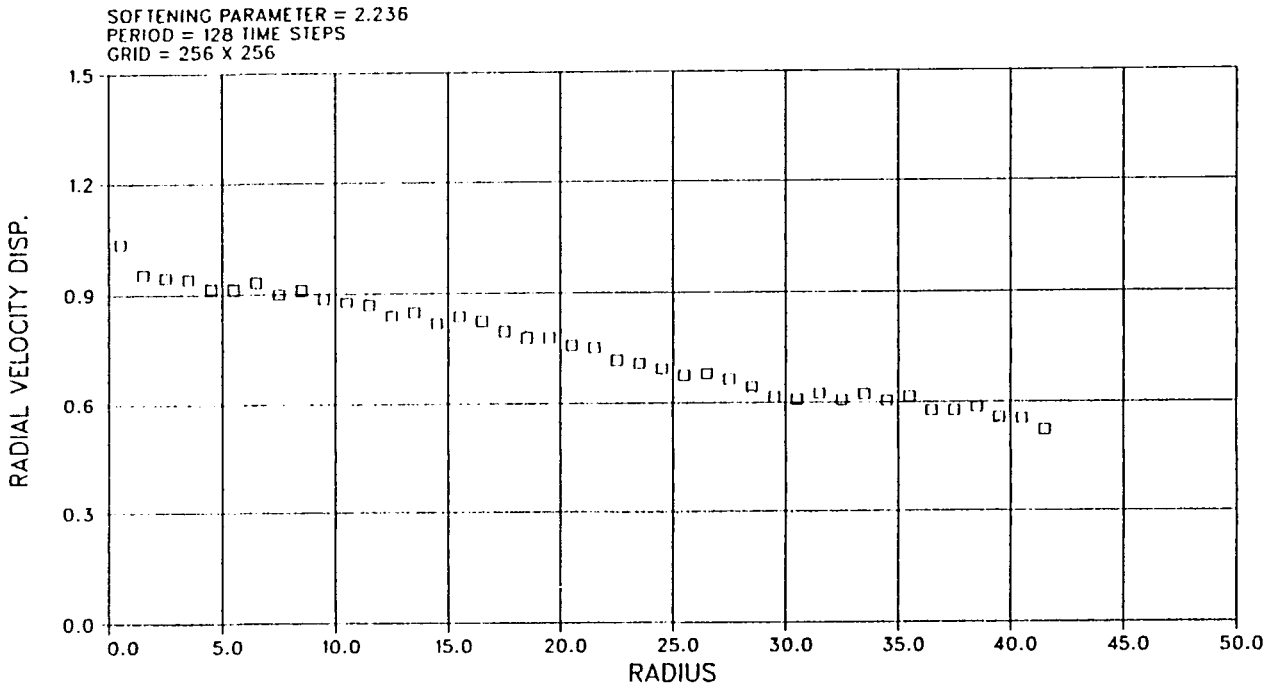

Fig 6-19: Radial velocity dispersion for the $\Omega = 0.4\Omega_0$ Kalnajs disk as calculated by CART2D for period = 5. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–20
VELOCITY DISPERSION: PERIOD = 6: OMEGA = 0.4

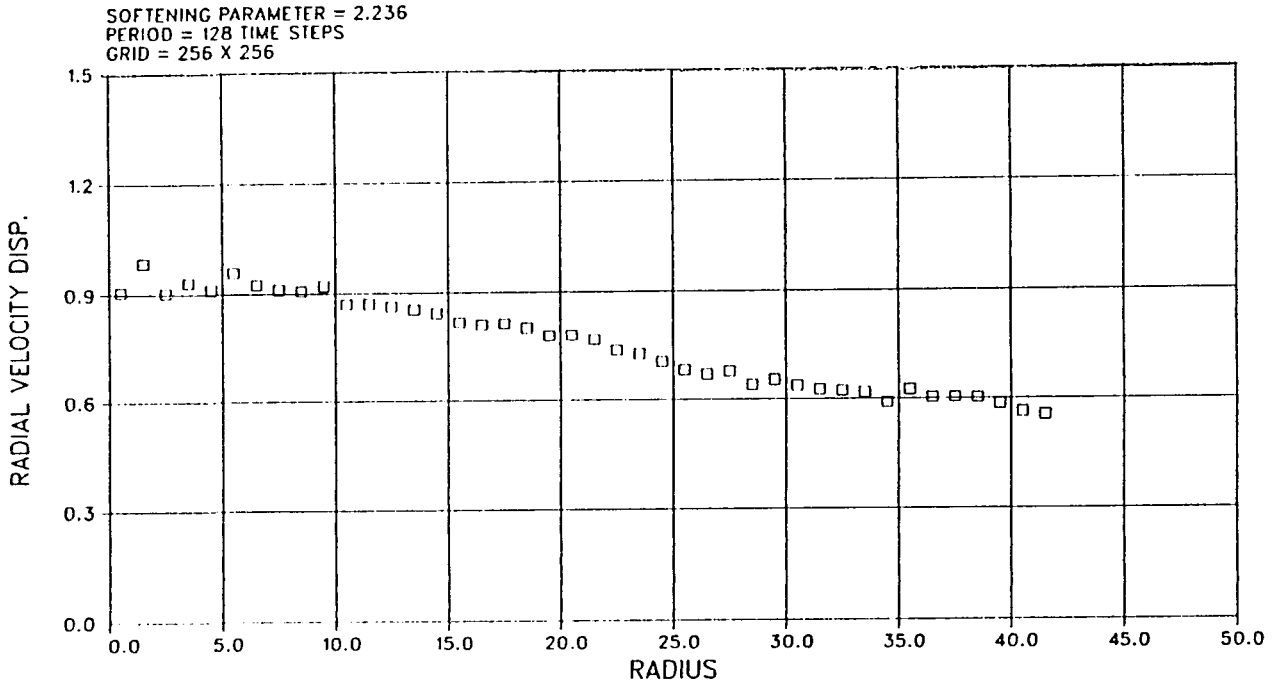

Fig 6-20: Radial velocity dispersion for the $\Omega = 0.4\Omega_0$ Kalnajs disk as calculated by CART2D for period = 6. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–21
SURFACE DENSITY: PERIOD = 0: OMEGA = 0.4

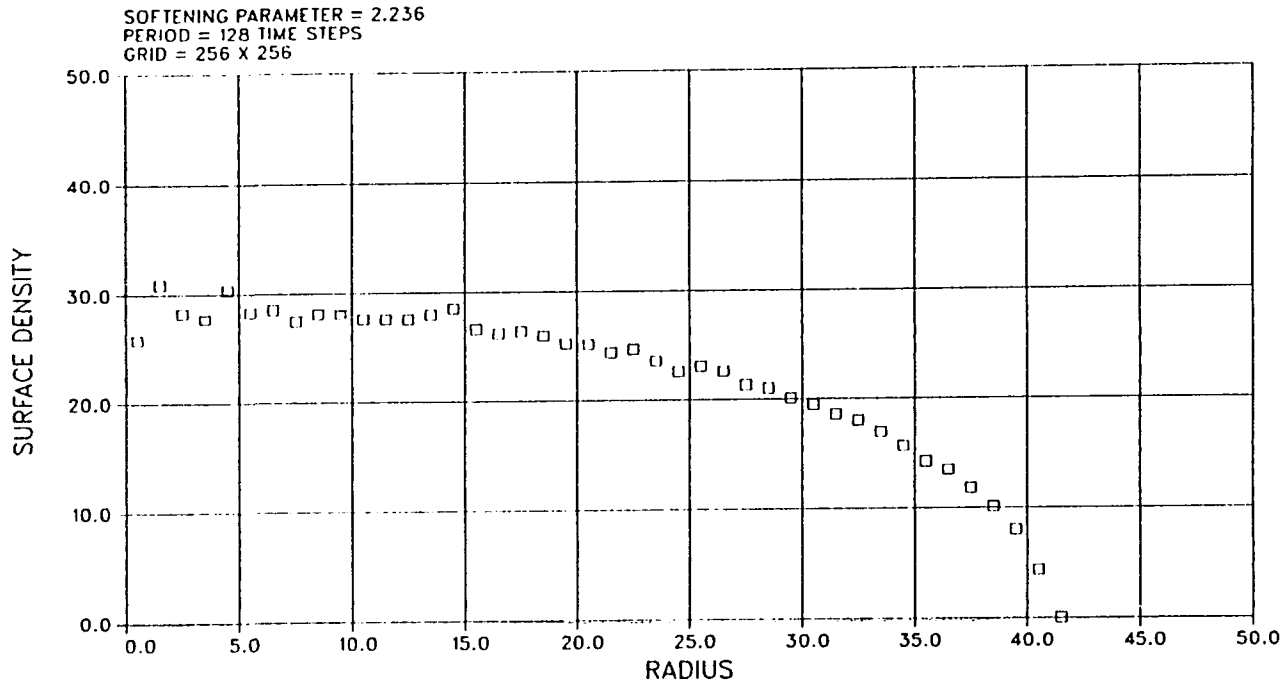

Fig 6-21: Surface density for the $\Omega = 0.4\Omega_0$ Kalnajs disk as calculated by CART2D for period = 0. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–22
SURFACE DENSITY: PERIOD = 1: OMEGA = 0.4

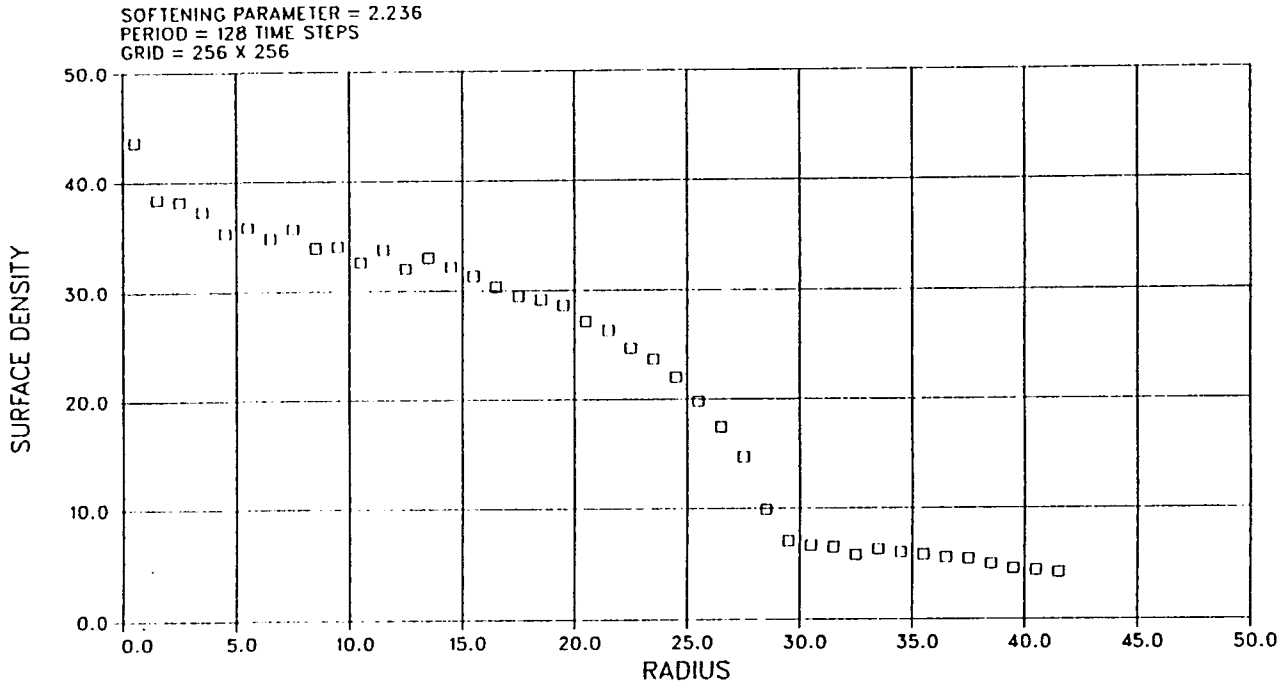

Fig 6-22: Surface density for the $\Omega = 0.4\Omega_0$ Kalnajs disk as calculated by CART2D for period = 1. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–23
SURFACE DENSITY: PERIOD = 2: OMEGA = 0.4

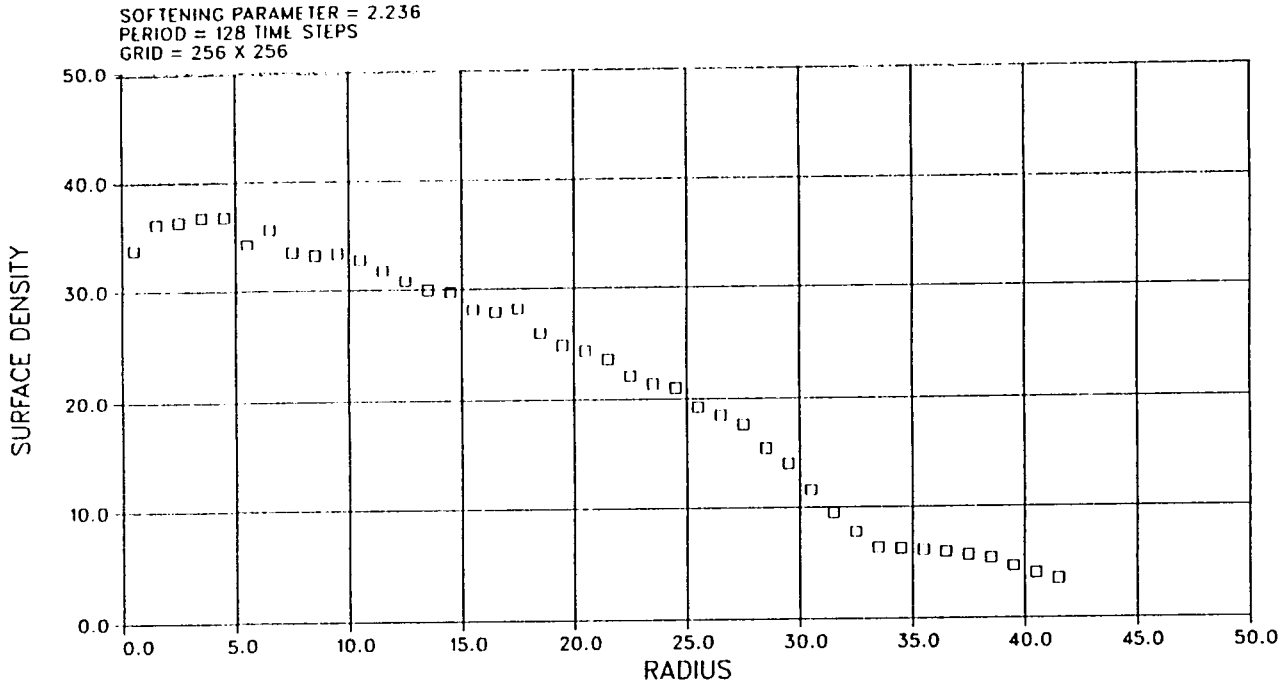

Fig 6-23: Surface density for the $\Omega = 0.4\Omega_0$ Kalnajs disk as calculated by CART2D for period = 2. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–24
SURFACE DENSITY: PERIOD = 3: OMEGA = 0.4

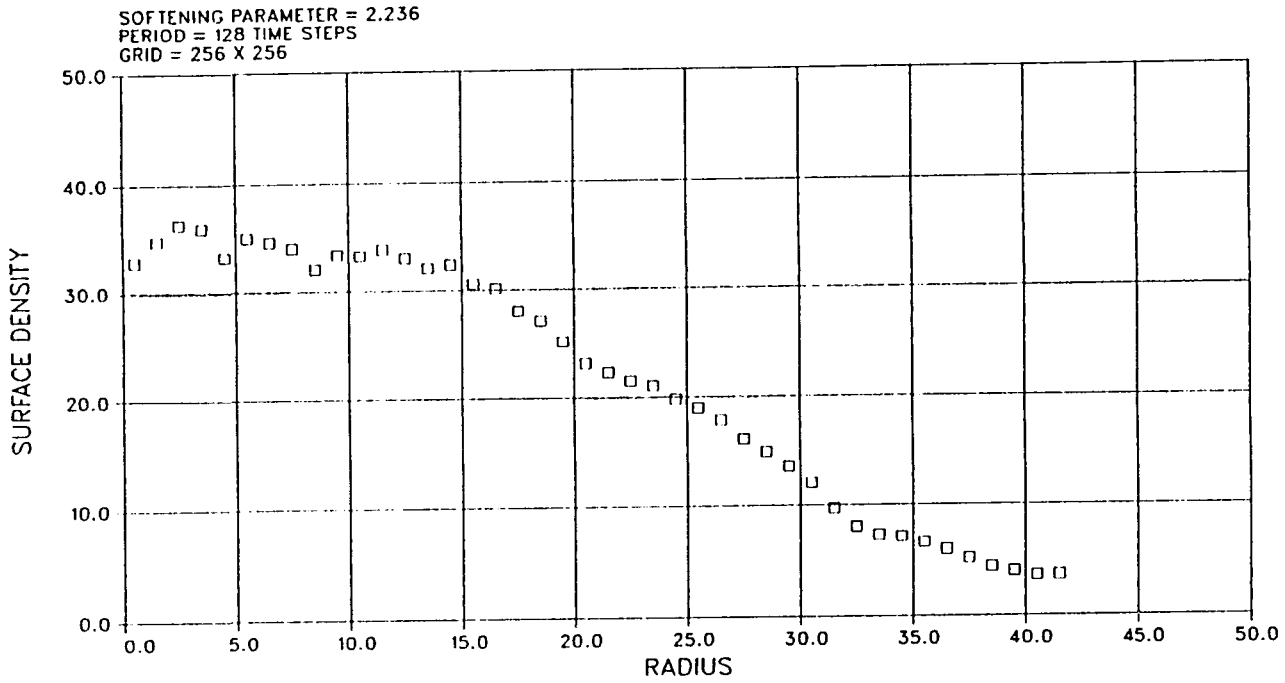

Fig 6-24: Surface density for the $\Omega = 0.4\Omega_0$ Kalnajs disk as calculated by CART2D for period = 3. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–25
SURFACE DENSITY: PERIOD = 4: OMEGA = 0.4

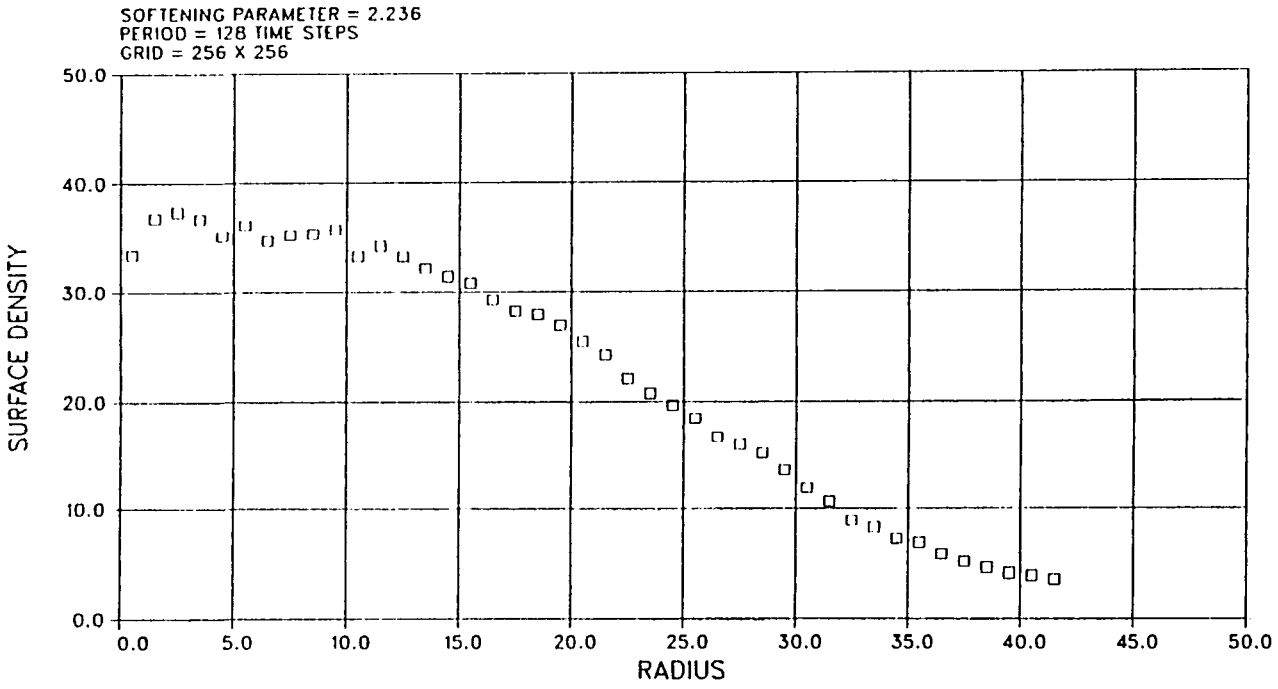

Fig 6-25: Surface density for the $\Omega = 0.4\Omega_0$ Kalnajs disk as calculated by CART2D for period = 4. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–26
SURFACE DENSITY: PERIOD = 5: OMEGA = 0.4

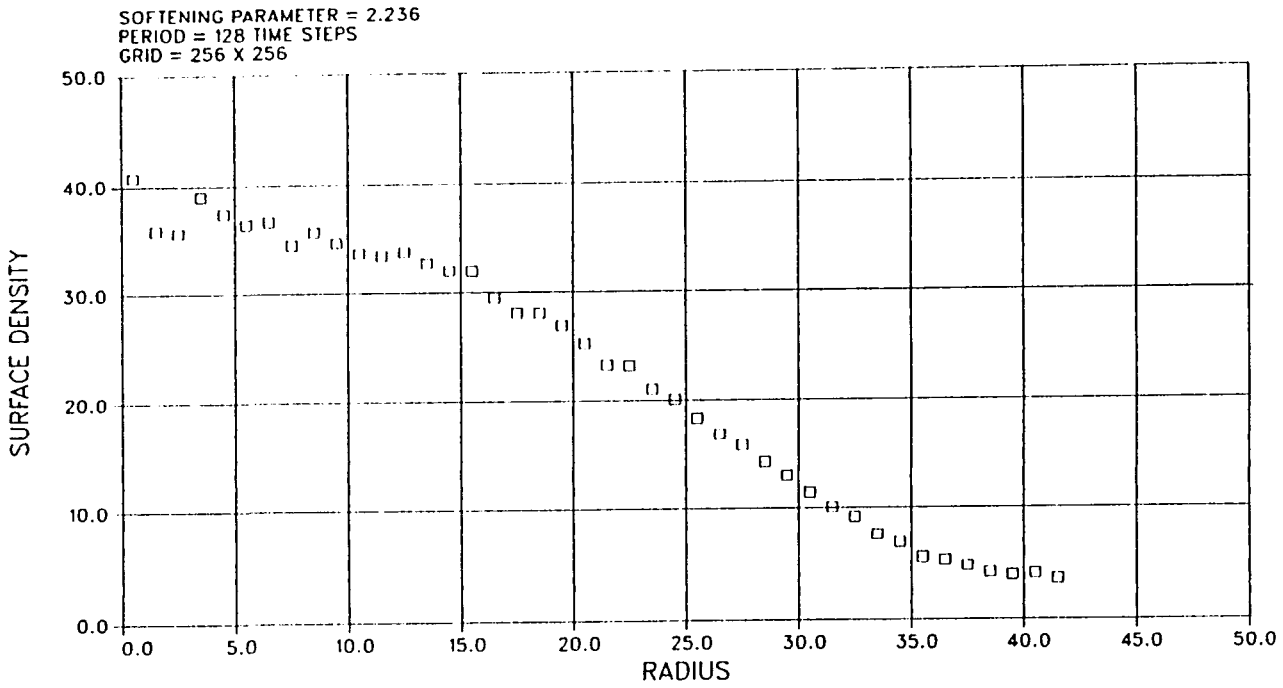

Fig 6-26: Surface density for the $\Omega = 0.4\Omega_0$ Kalnajs disk as calculated by CART2D for period = 5. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–27
SURFACE DENSITY: PERIOD = 6: OMEGA = 0.4

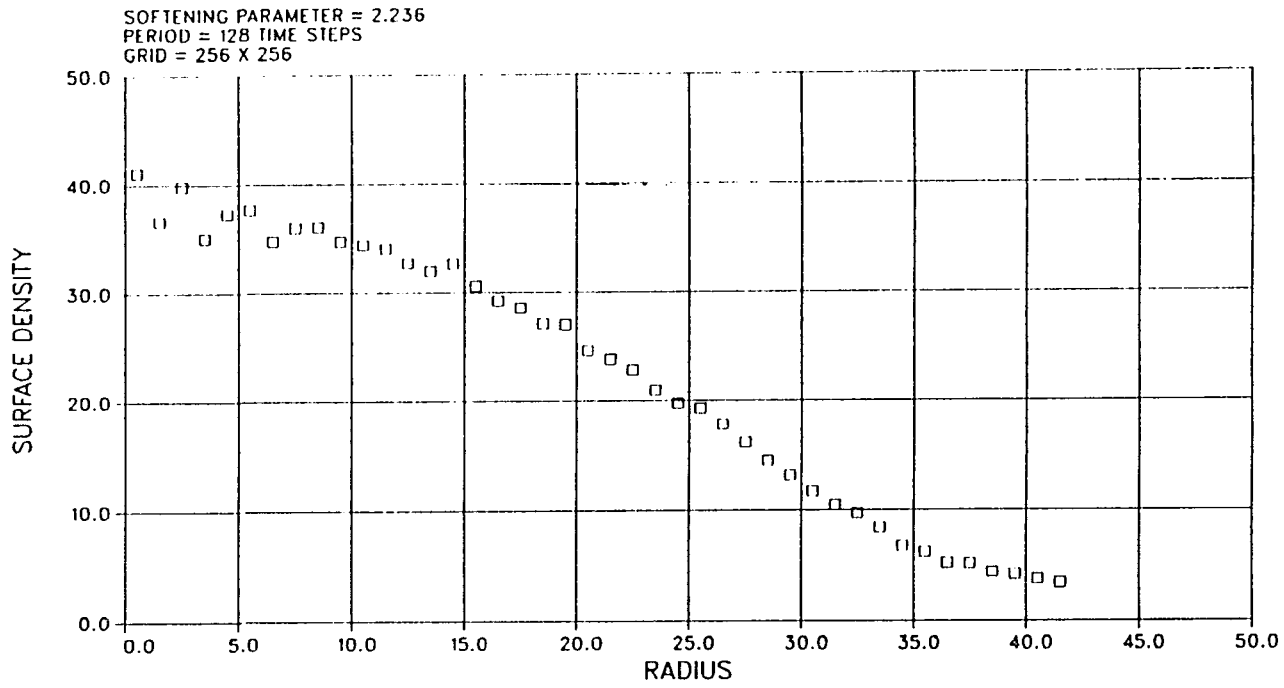

Fig 6-27: Surface density for the $\Omega = 0.4\Omega_0$ Kalnajs disk as calculated by CART2D for period = 6. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–28
Q: PERIOD = 0: OMEGA = 0.4

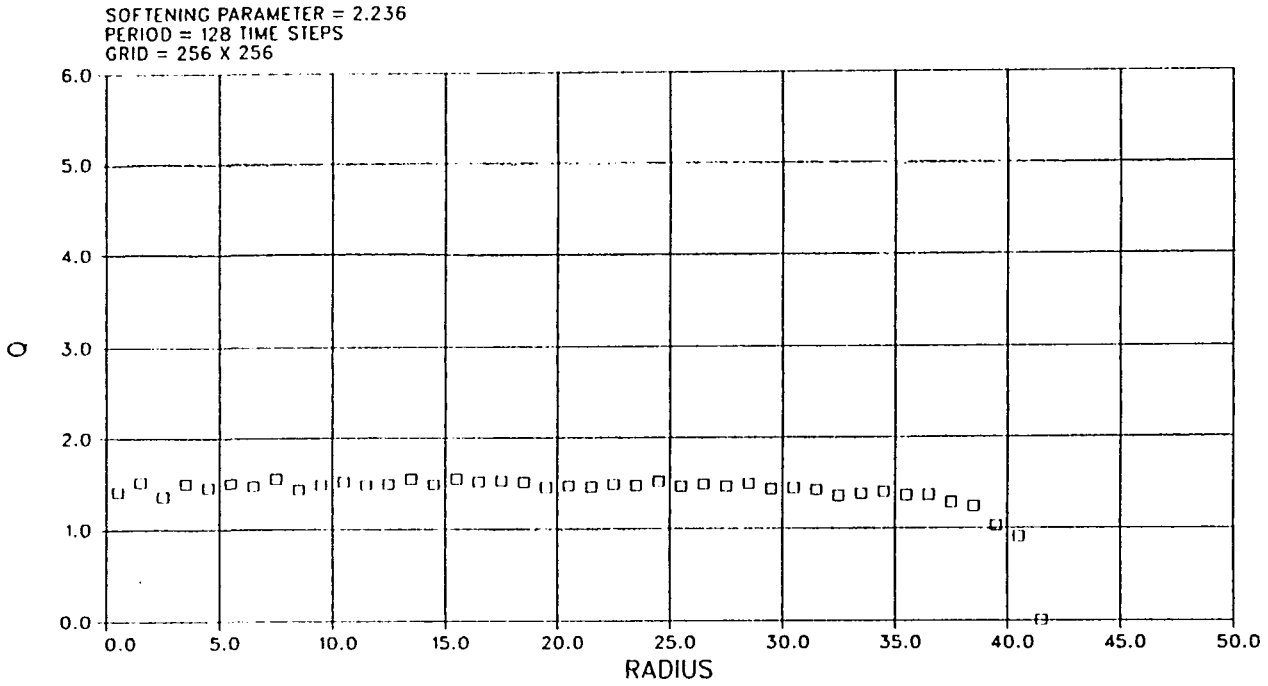

Fig 6-28: Q values for the $\Omega = 0.4\Omega_0$ Kalnajs disk for period = 0. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–29
Q: PERIOD = 1: OMEGA = 0.4

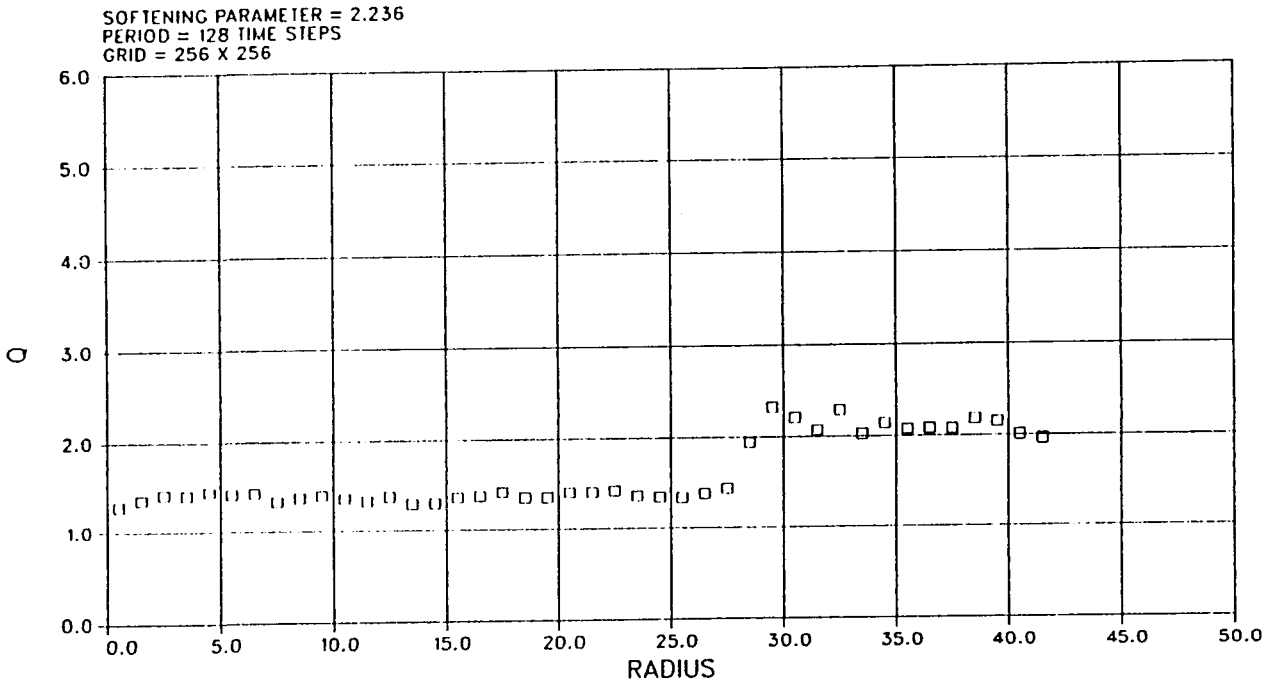

Fig 6-29: Q values for the $\Omega = 0.4\Omega_0$ Kalnajs disk for period = 1. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–30
Q: PERIOD = 2: OMEGA = 0.4

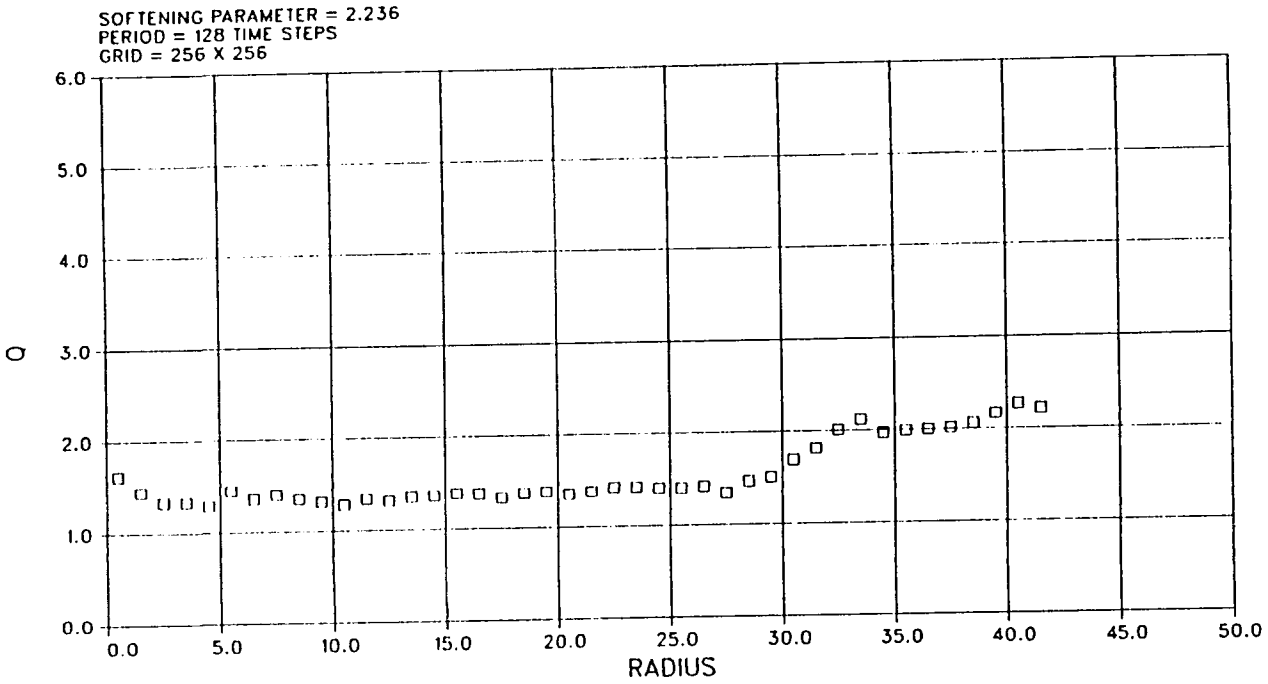

Fig 6-30: Q values for the $\Omega = 0.4\Omega_0$ Kalnajs disk for period = 2. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–31
Q: PERIOD = 3: OMEGA = 0.4

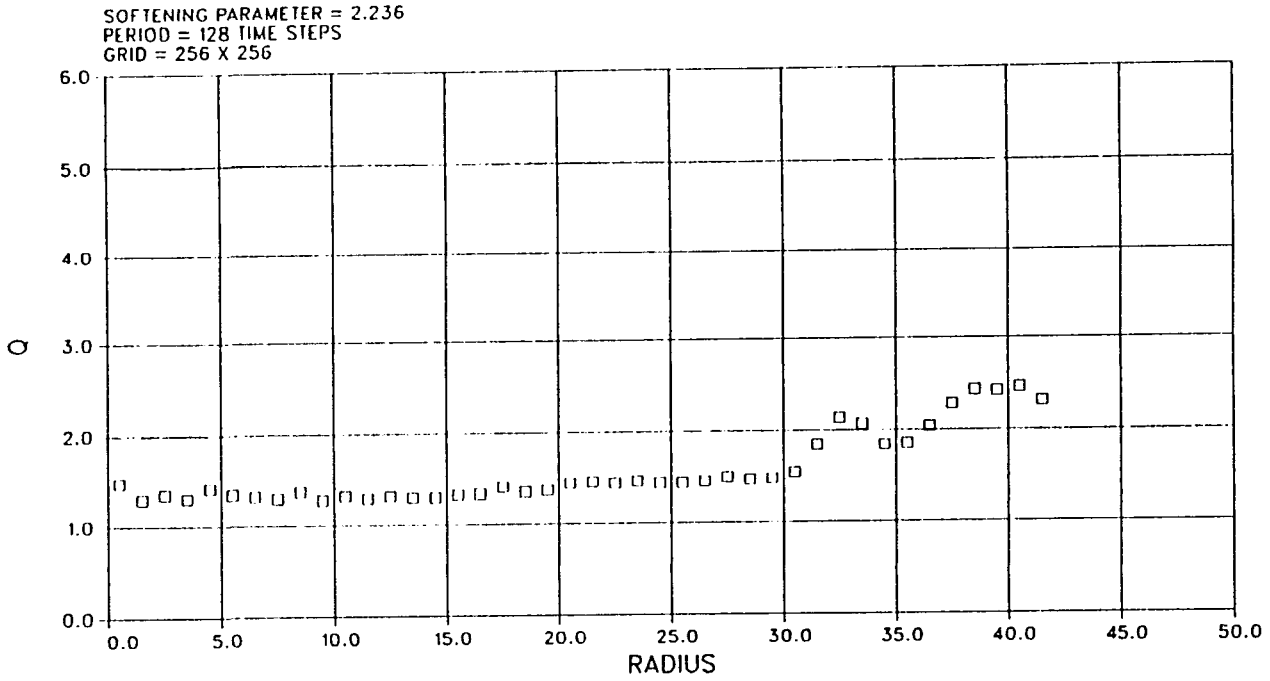

Fig 6-31: Q values for the $\Omega = 0.4\Omega_0$ Kalnajs disk for period = 3. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–32
Q: PERIOD = 4: OMEGA = 0.4

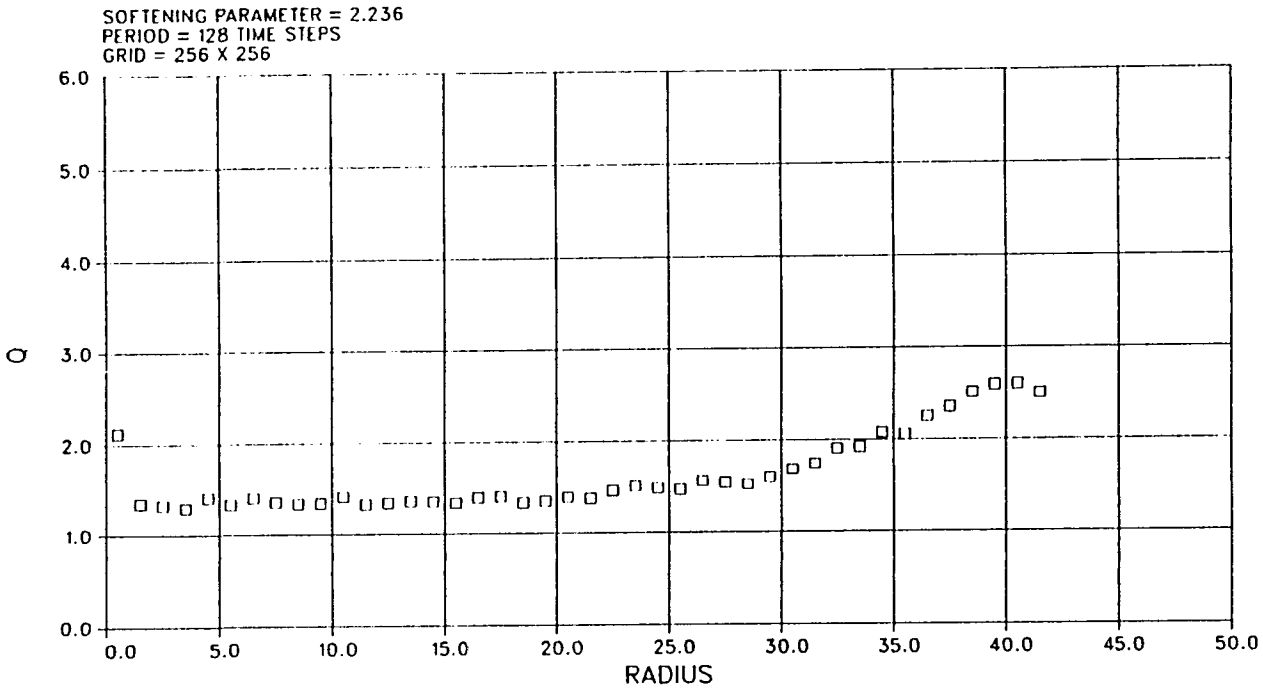

Fig 6-32: Q values for the $\Omega = 0.4\Omega_0$ Kalnajs disk for period = 4. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6-33
Q: PERIOD = 5: OMEGA = 0.4

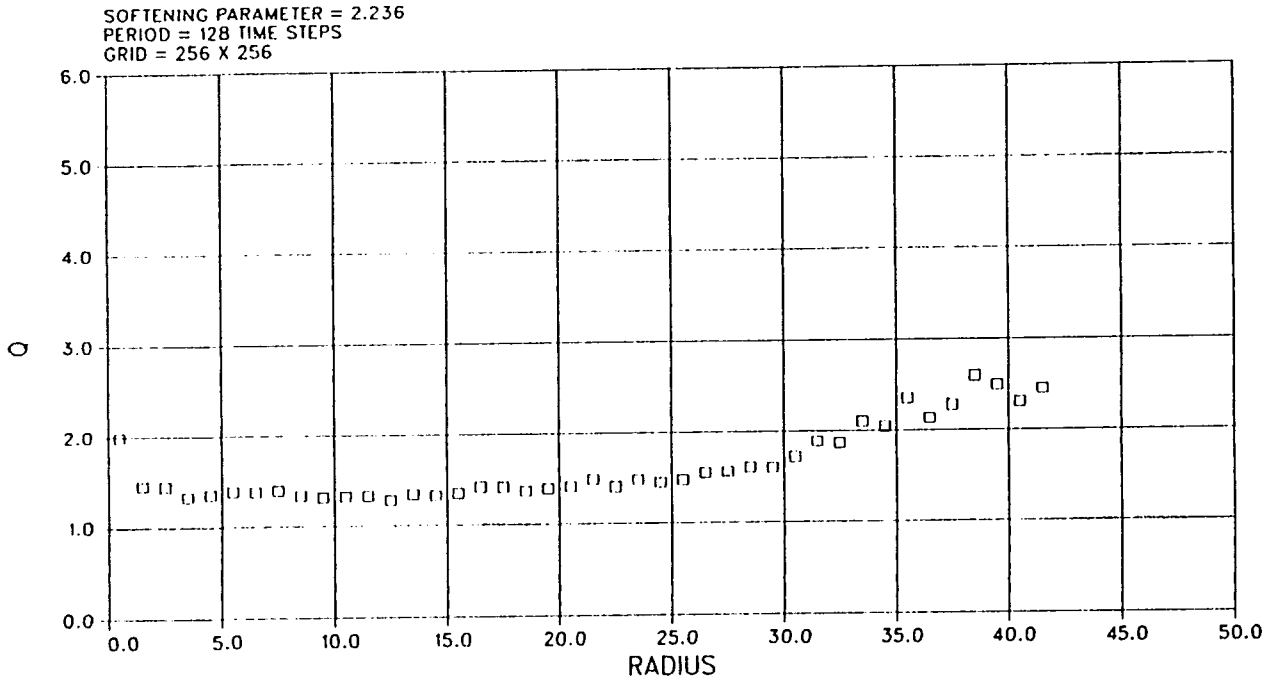

Fig 6-33: Q values for the $\Omega = 0.4\Omega_0$ Kalnajs disk for period = 5. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–34
Q: PERIOD = 6: OMEGA = 0.4

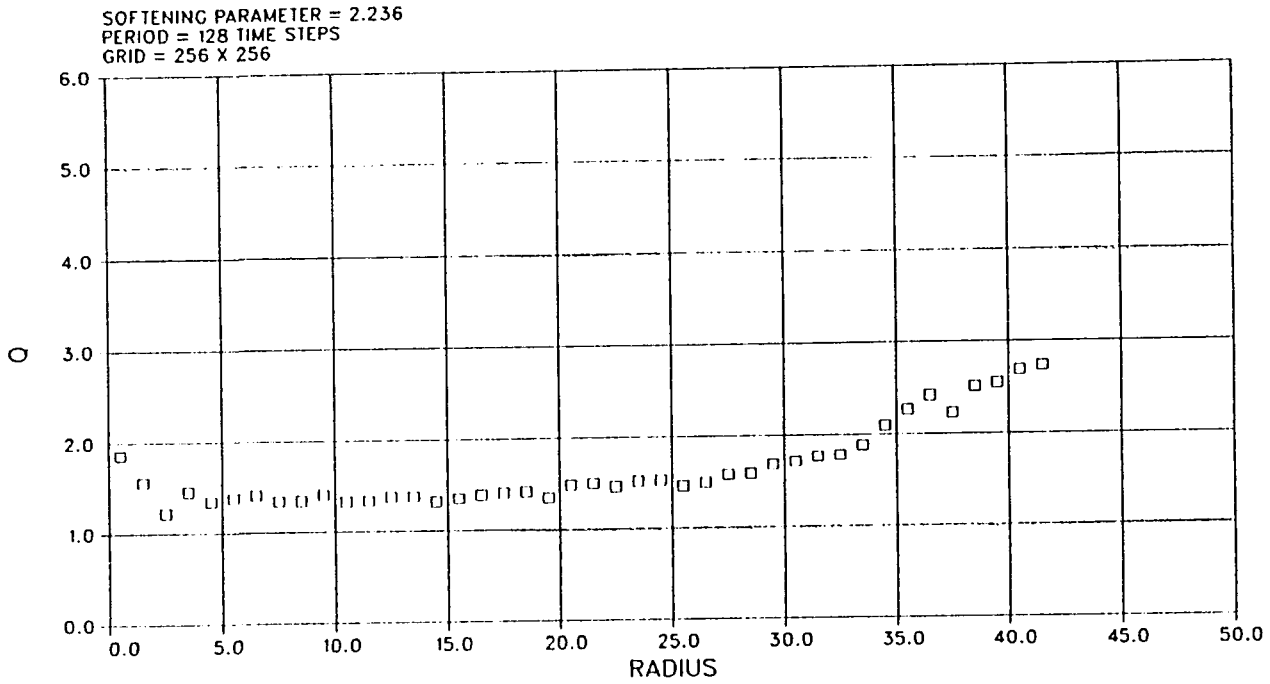

Fig 6-34: Q values for the $\Omega = 0.4\Omega_0$ Kalnajs disk for period = 6. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–35
PERCENT PARTICLE SPILLS: OMEGA = 0.4

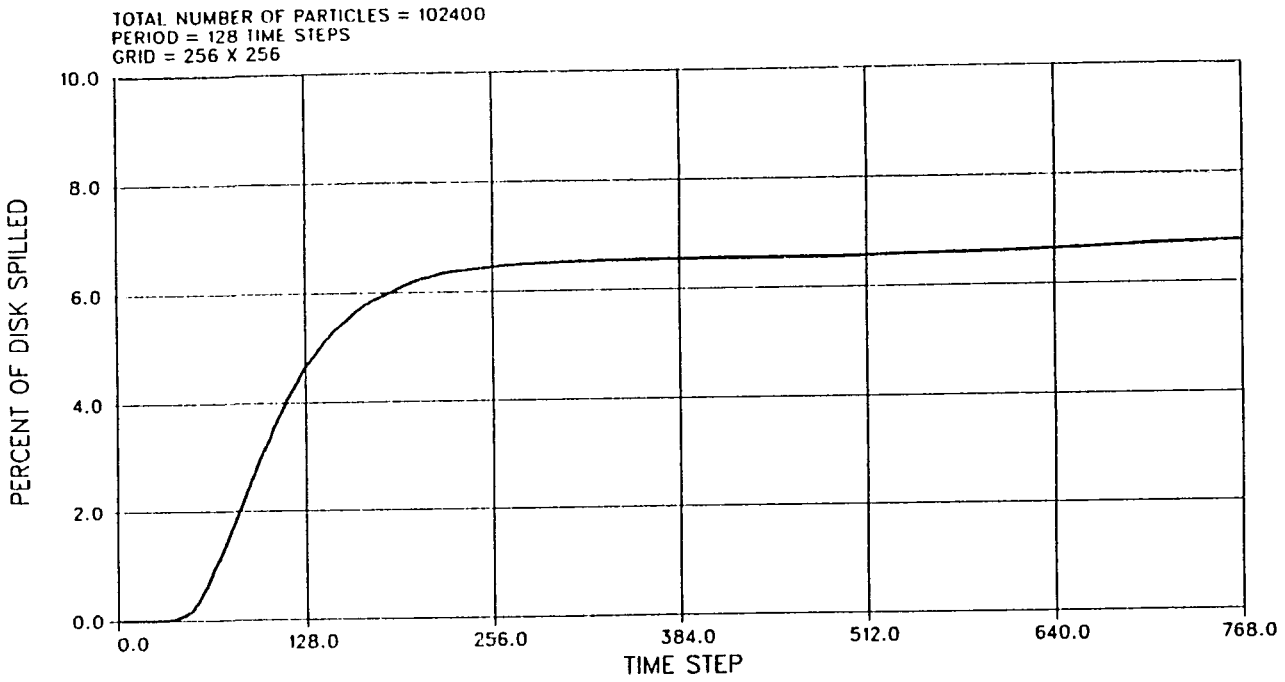

Fig 6-35: Percent of the total disk that has spilled as a function of time step. Each period is 128.8 time steps.

FIGURE 6–36
PERIOD = 0: OMEGA = 0.8

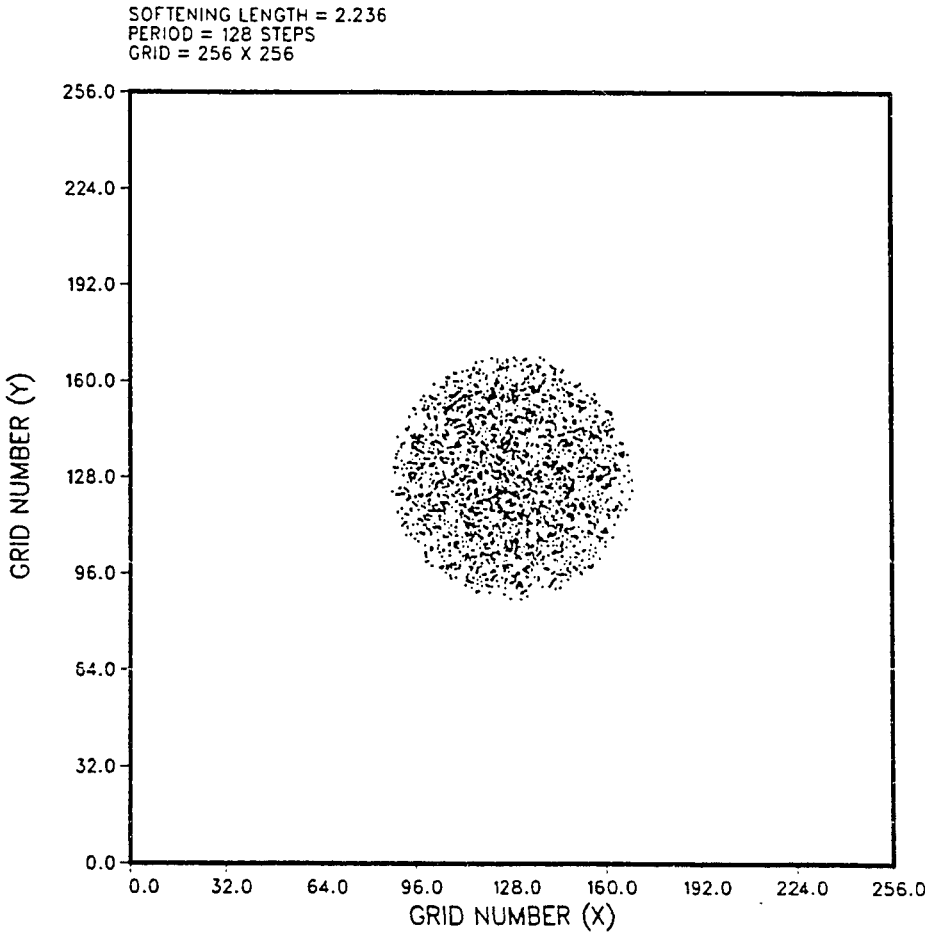

Fig 6-36: Particle plot for the $\Omega = 0.8\Omega_0$ Kalnajs disk omega model. This figure shows the initial load.

FIGURE 6–37
PERIOD = 1: OMEGA = 0.8

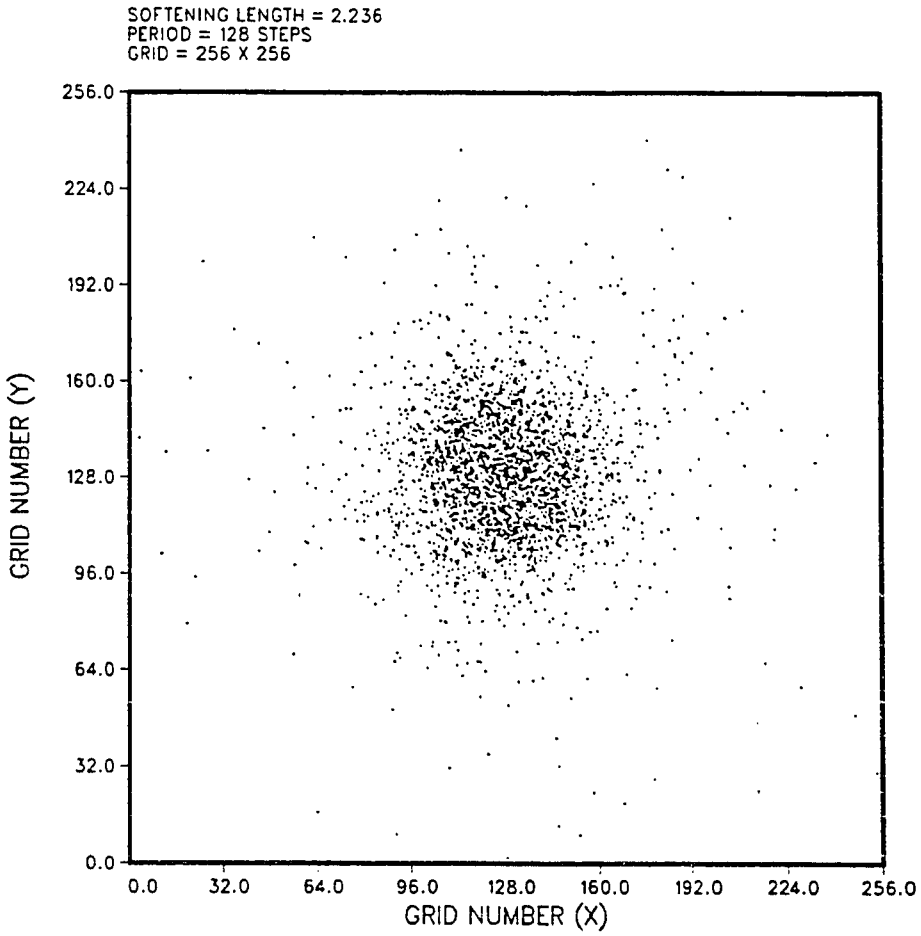

Fig 6-37: Particle plot for the $\Omega = 0.8\Omega_0$ Kalnajs disk omega model. This figure shows the disk after one period of revolution. One period is 128.8 time steps. The value of softening is $\epsilon^2 = 5$. The computational grid is 256 X 256, and 1/40th of the particles used are plotted.

FIGURE 6–38
PERIOD = 2: OMEGA = 0.8

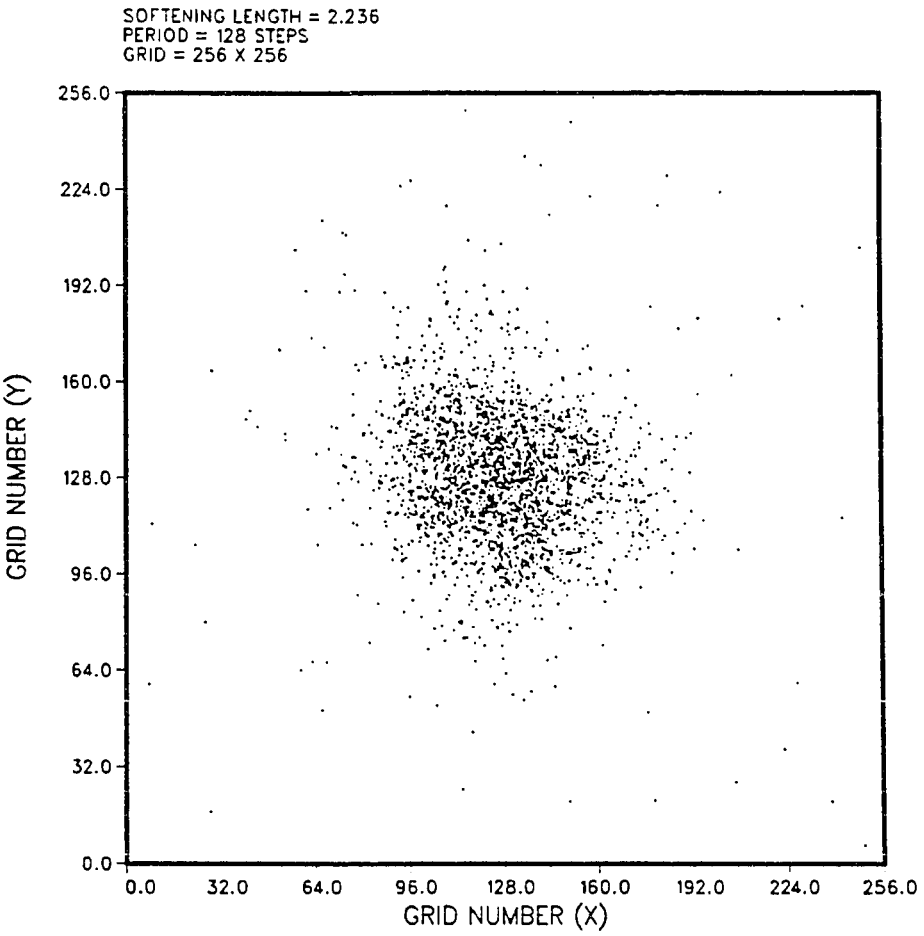

Fig 6-38: Particle plot for the $\Omega = 0.8\Omega_0$ Kalnajs disk omega model. This figure shows the disk after two periods of revolution. One period is 128.8 time steps. The value of softening is $\epsilon^2 = 5$. The computational grid is 256 X 256, and 1/40th of the particles used are plotted.

FIGURE 6–39
PERIOD = 3: OMEGA = 0.8

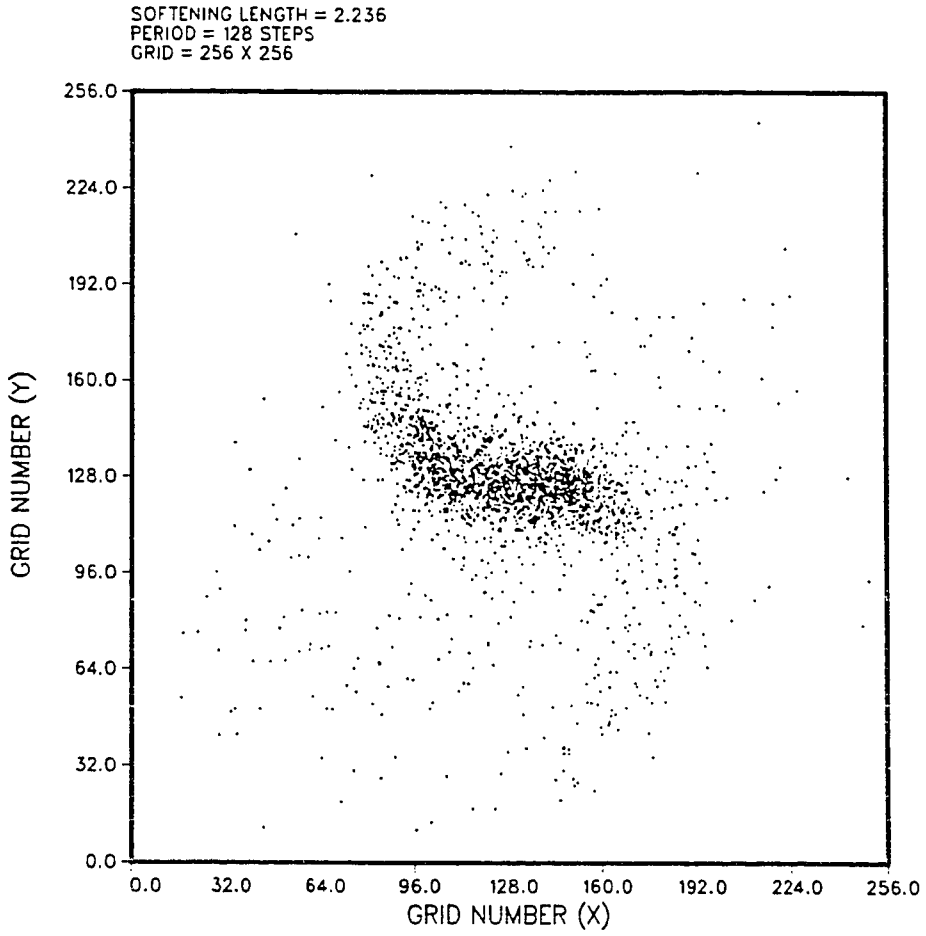

Fig 6-39: Particle plot for the $\Omega = 0.8\Omega_0$ Kalnajs disk omega model. This figure shows the disk after three periods of revolution. One period is 128.8 time steps. The value of softening is $\epsilon^2 = 5$. The computational grid is 256 X 256, and 1/40th of the particles used are plotted.

FIGURE 6-40
PERIOD = 4: OMEGA = 0.8

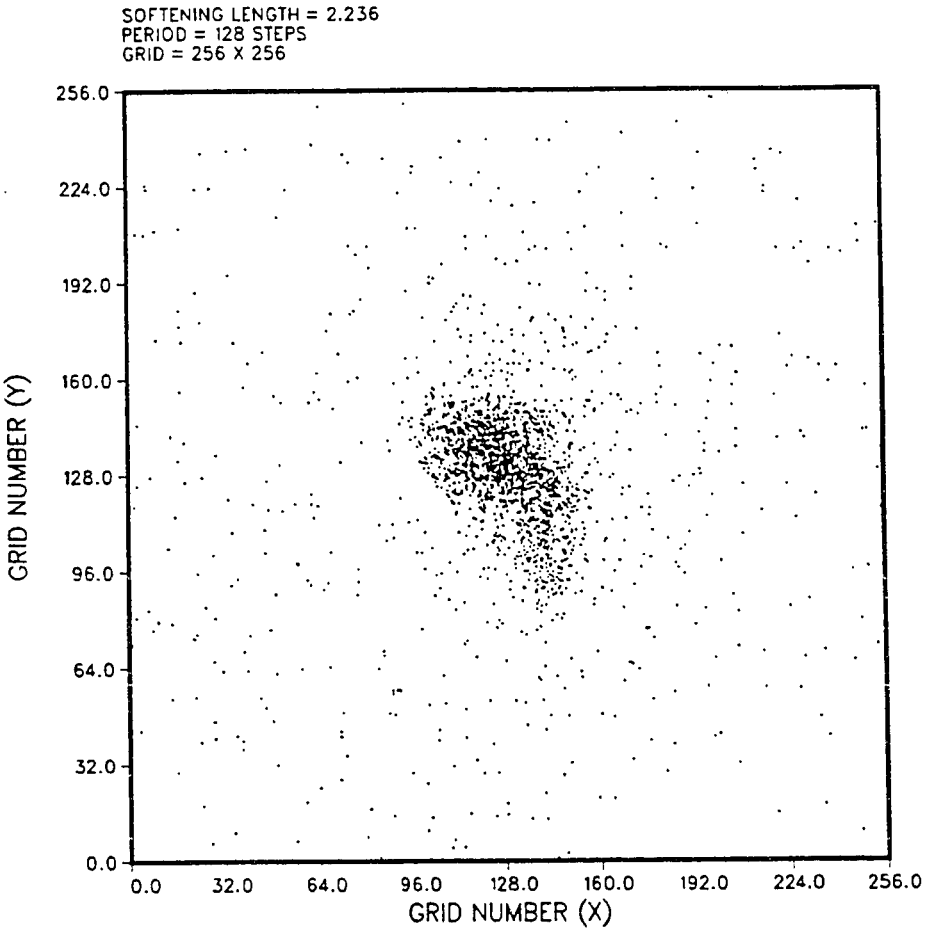

Fig 6-40: Particle plot for the $\Omega = 0.8\Omega_0$ Kalnajs disk omega model. This figure shows the disk after four periods of revolution. One period is 128.8 time steps. The value of softening is $\epsilon^2 = 5$. The computational grid is 256 X 256, and 1/40th of the particles used are plotted.

FIGURE 6–41
VELOCITY DISPERSION: PERIOD = 0: OMEGA = 0.8

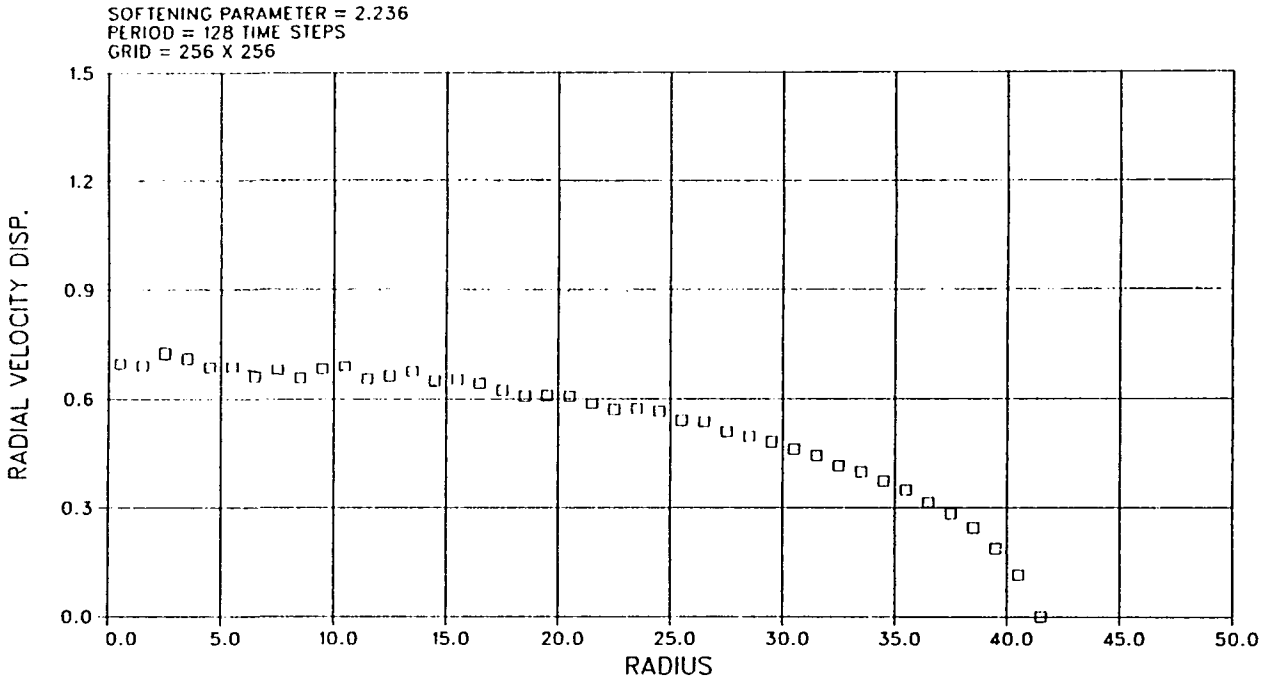

Fig 6-41: Radial velocity dispersion for the $\Omega = 0.8\Omega_0$ Kalnajs disk as calculated by CART2D for period = 0. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–42
VELOCITY DISPERSION: PERIOD = 1: OMEGA = 0.8

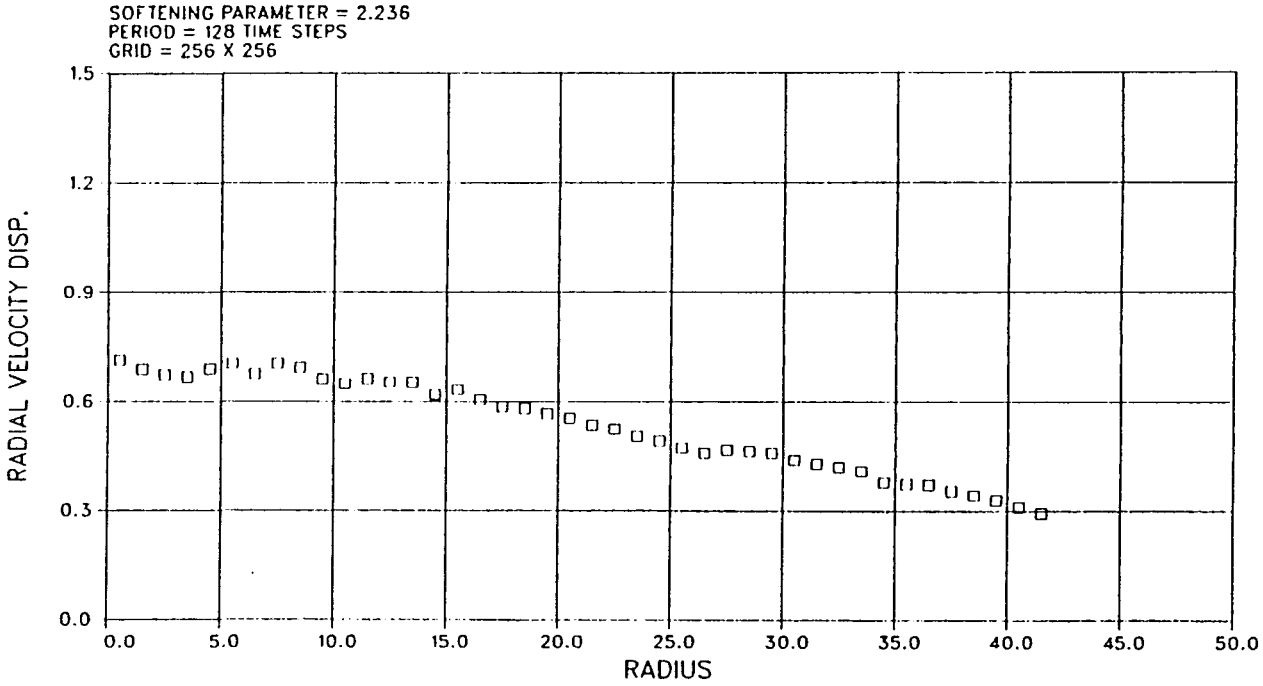

Fig 6-42: Radial velocity dispersion for the $\Omega = 0.8\Omega_0$ Kalnajs disk as calculated by CART2D for period = 1. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–43
VELOCITY DISPERSION: PERIOD = 2: OMEGA = 0.8

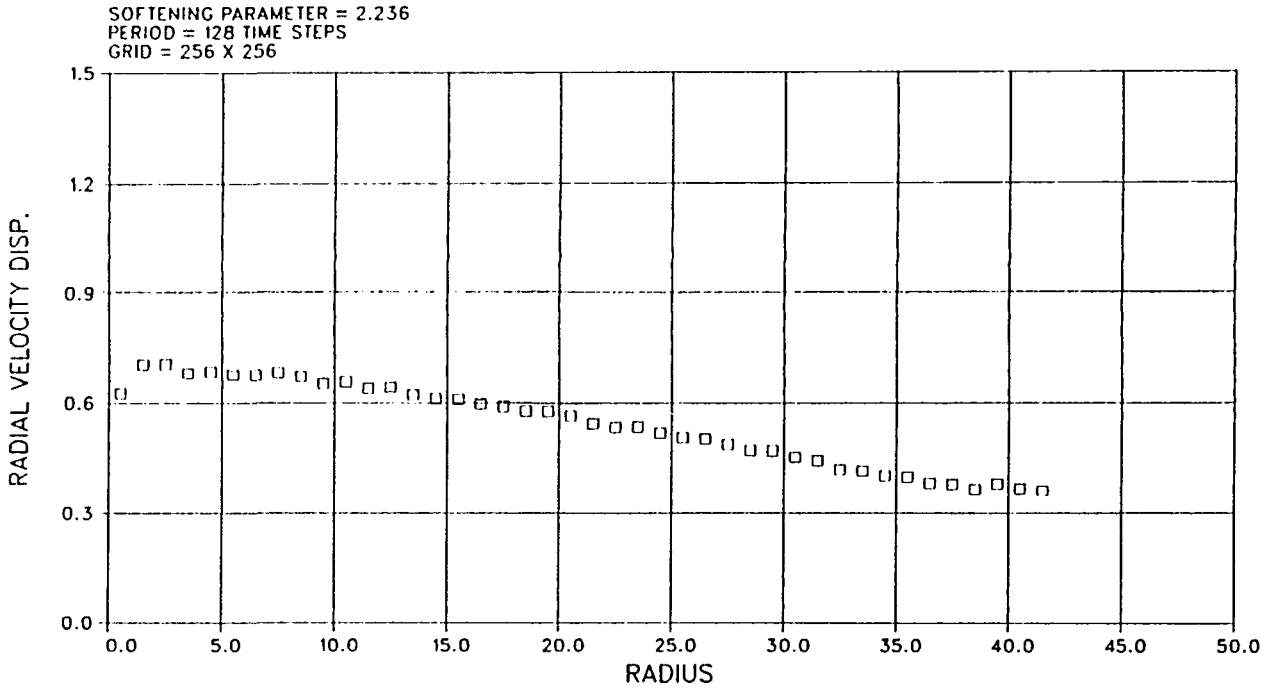

Fig 6-43: Radial velocity dispersion for the $\Omega = 0.8\Omega_0$ Kalnajs disk as calculated by CART2D for period = 2. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–44
VELOCITY DISPERSION: PERIOD = 3: OMEGA = 0.8

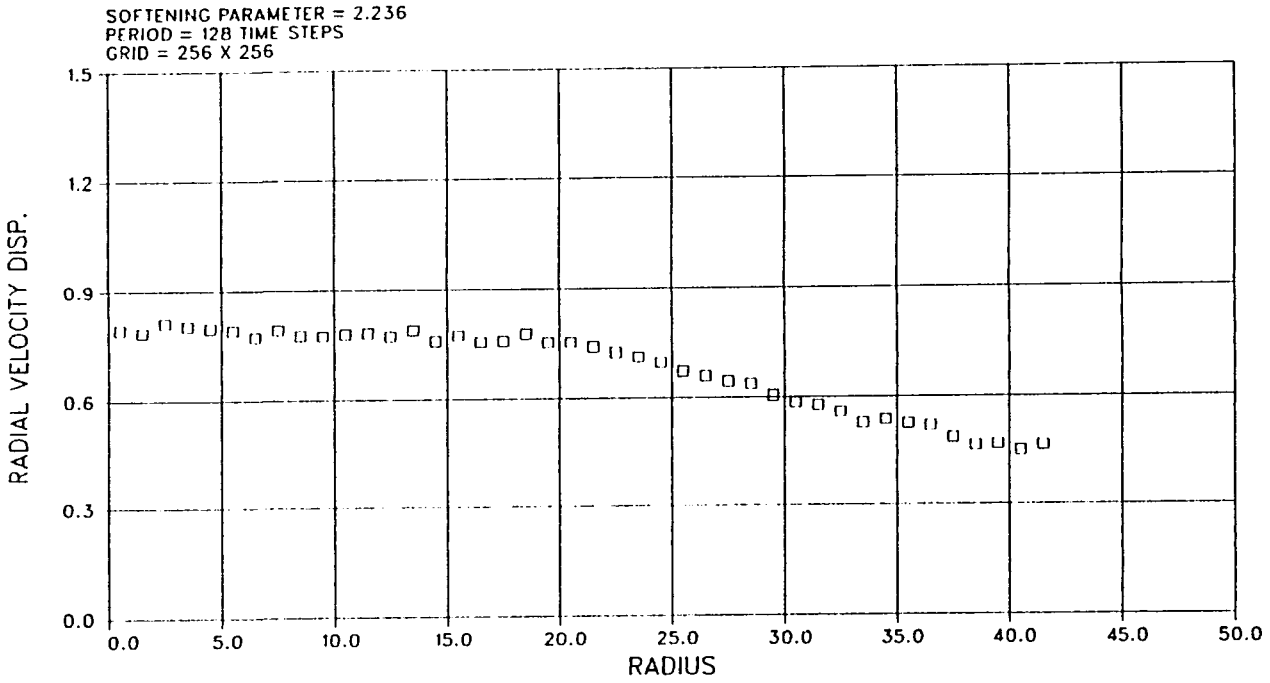

Fig 6-44: Radial velocity dispersion for the $\Omega = 0.8\Omega_0$ Kalnajs disk as calculated by CART2D for period = 3. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–45
VELOCITY DISPERSION: PERIOD = 4: OMEGA = 0.8

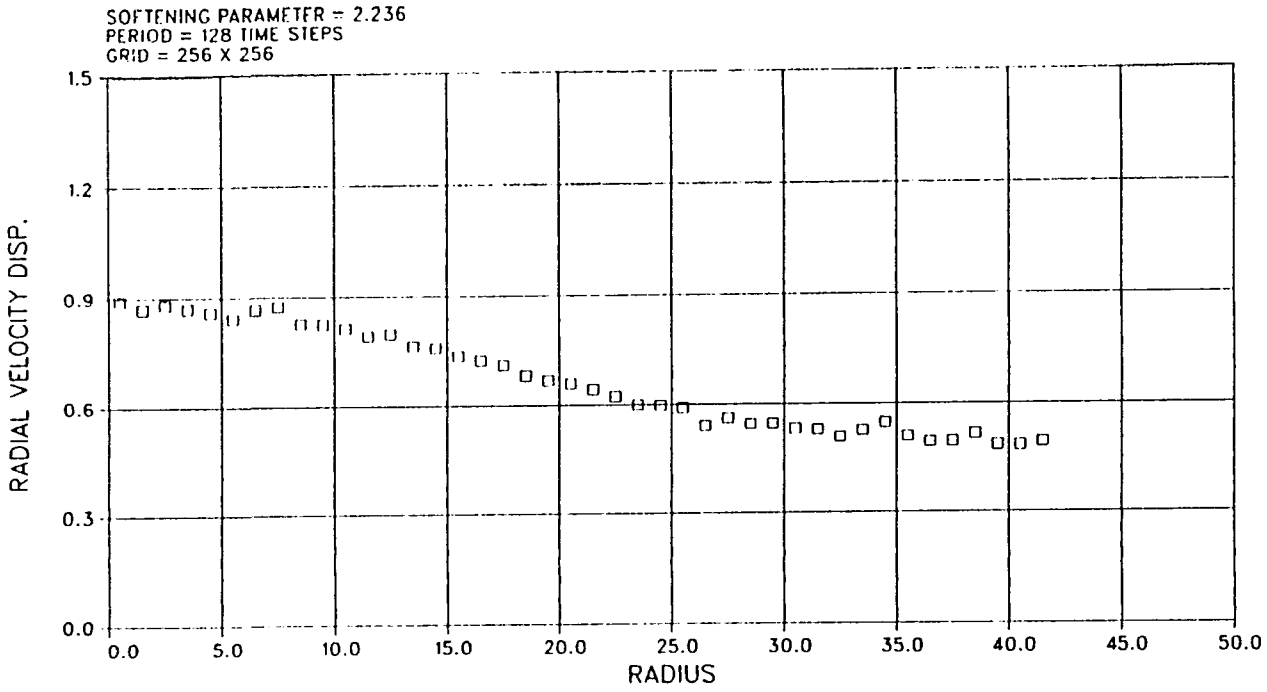

Fig 6-45: Radial velocity dispersion for the $\Omega = 0.8\Omega_0$ Kalnajs disk as calculated by CART2D for period = 4. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6-46
SURFACE DENSITY: PERIOD = 0: OMEGA = 0.8

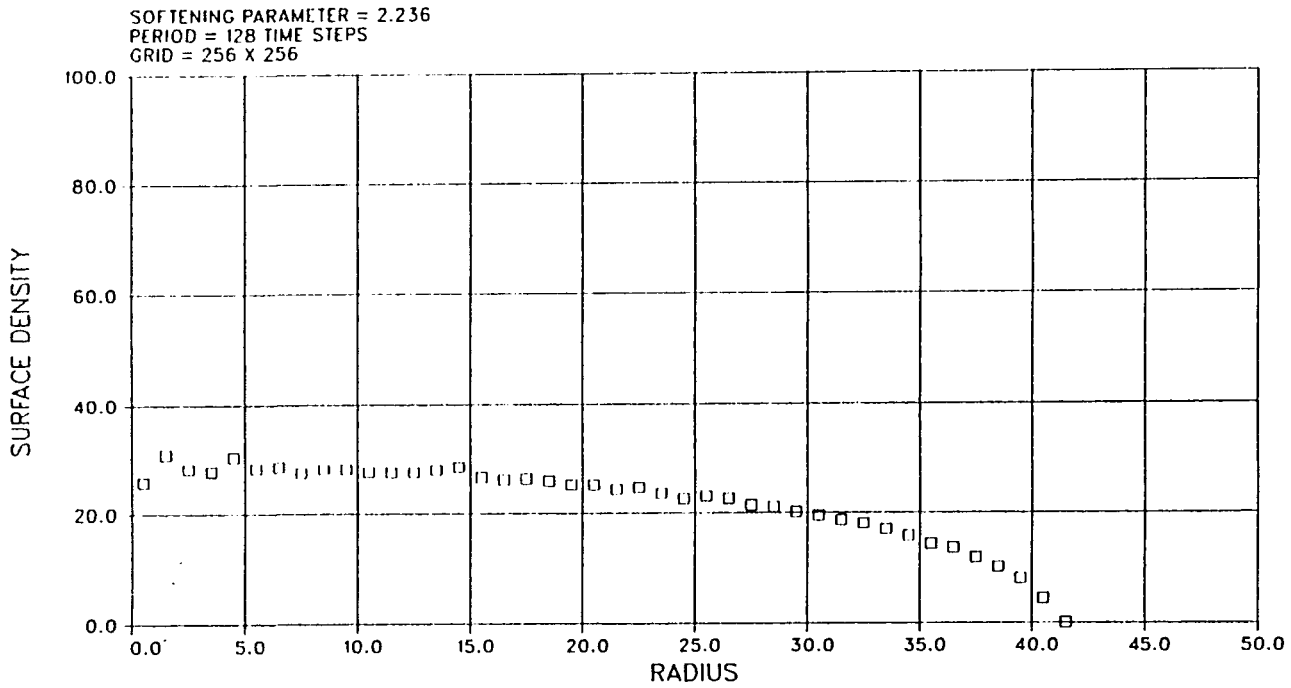

Fig 6-46: Surface density for the $\Omega = 0.8\Omega_0$ Kalnajs disk as calculated by CART2D for period = 0. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–47
SURFACE DENSITY: PERIOD = 1: OMEGA = 0.8

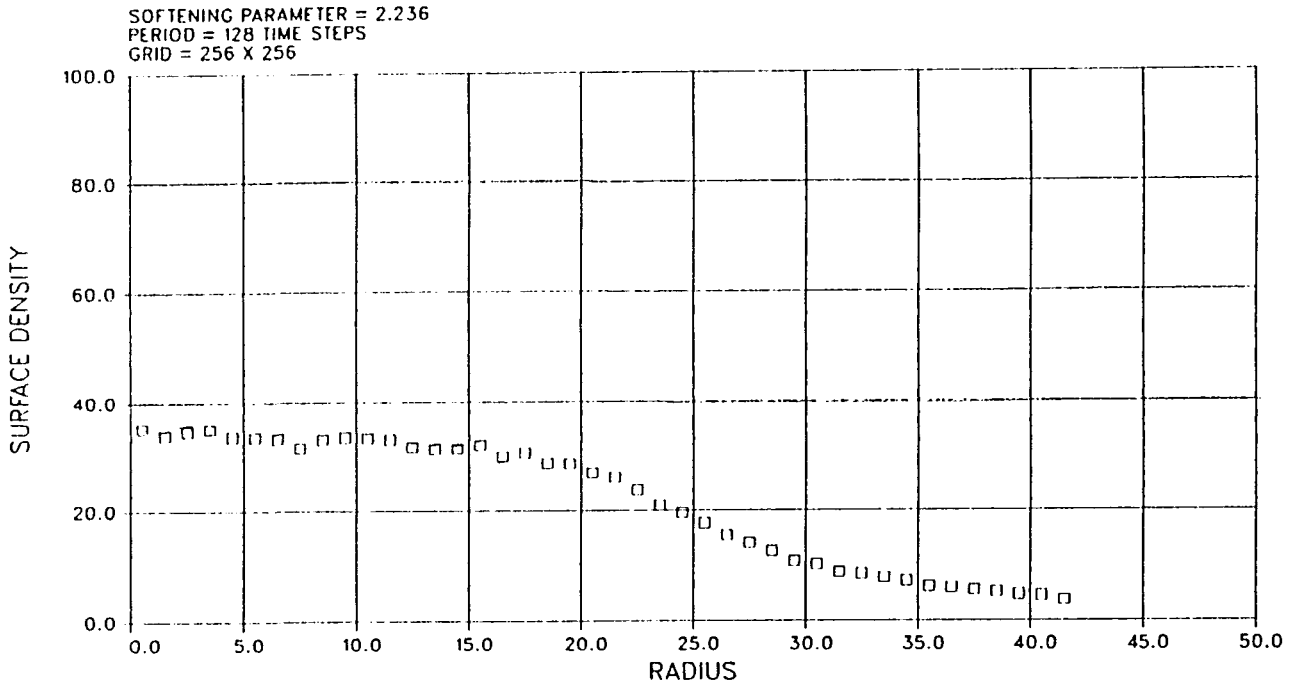

Fig 6-47: Surface density for the $\Omega = 0.8\Omega_0$ Kalnajs disk as calculated by CART2D for period = 1. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–48
SURFACE DENSITY: PERIOD = 2: OMEGA = 0.8

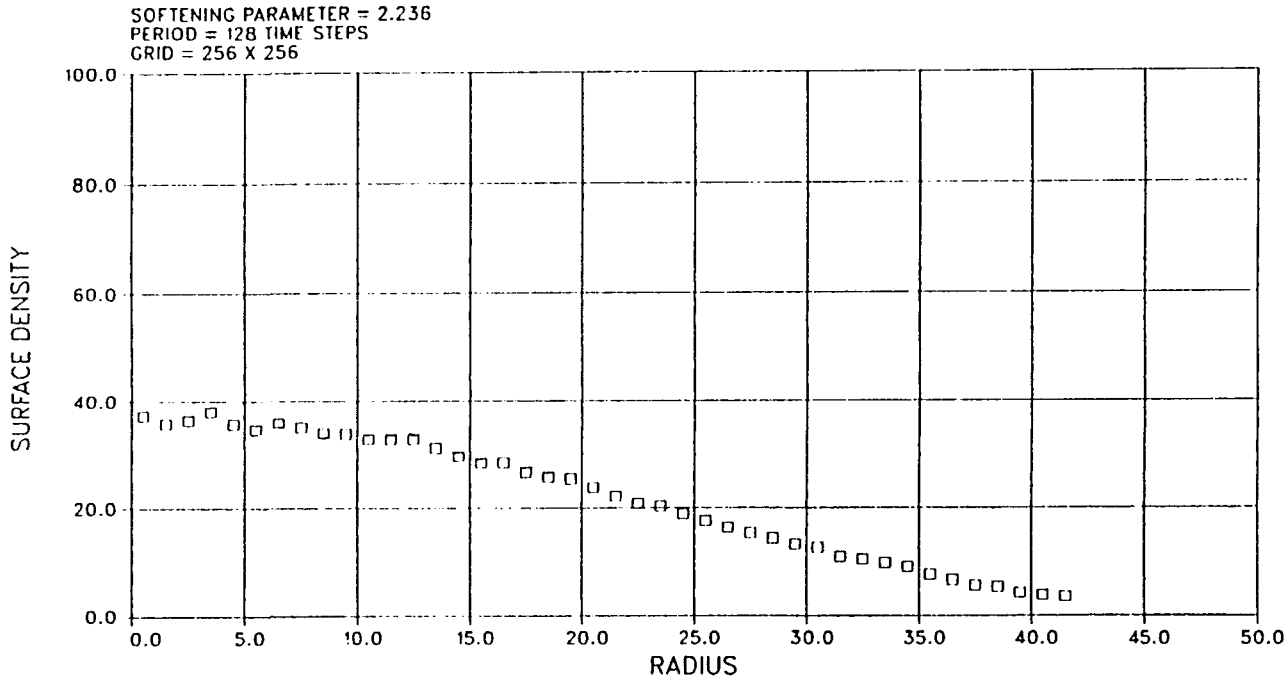

Fig 6-48: Surface density for the $\Omega = 0.8\Omega_0$ Kalnajs disk as calculated by CART2D for period = 2. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–49
SURFACE DENSITY: PERIOD = 3: OMEGA = 0.8

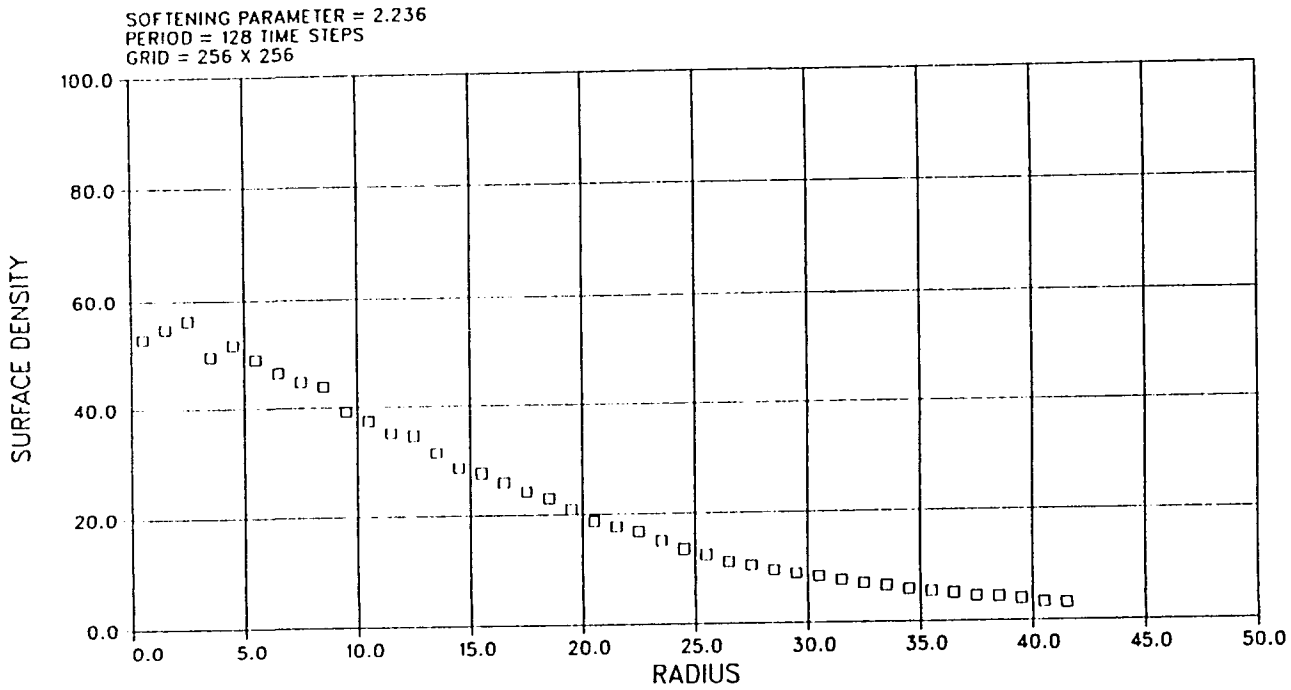

Fig 6-49: Surface density for the $\Omega = 0.8\Omega_0$ Kalnajs disk as calculated by CART2D for period = 3. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6-50
SURFACE DENSITY: PERIOD = 4: OMEGA = 0.8

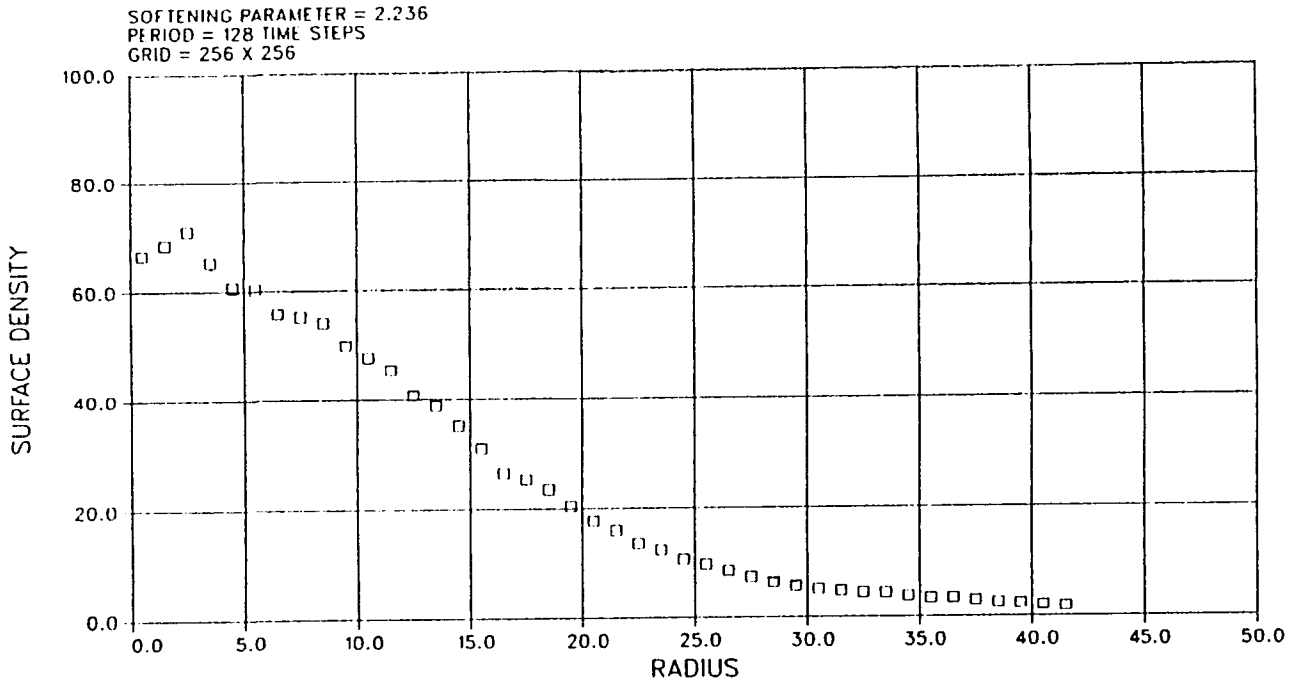

Fig 6-50: Surface density for the $\Omega = 0.8\Omega_0$ Kalnajs disk as calculated by CART2D for period = 4. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6-51
Q: PERIOD = 0: OMEGA = 0.8

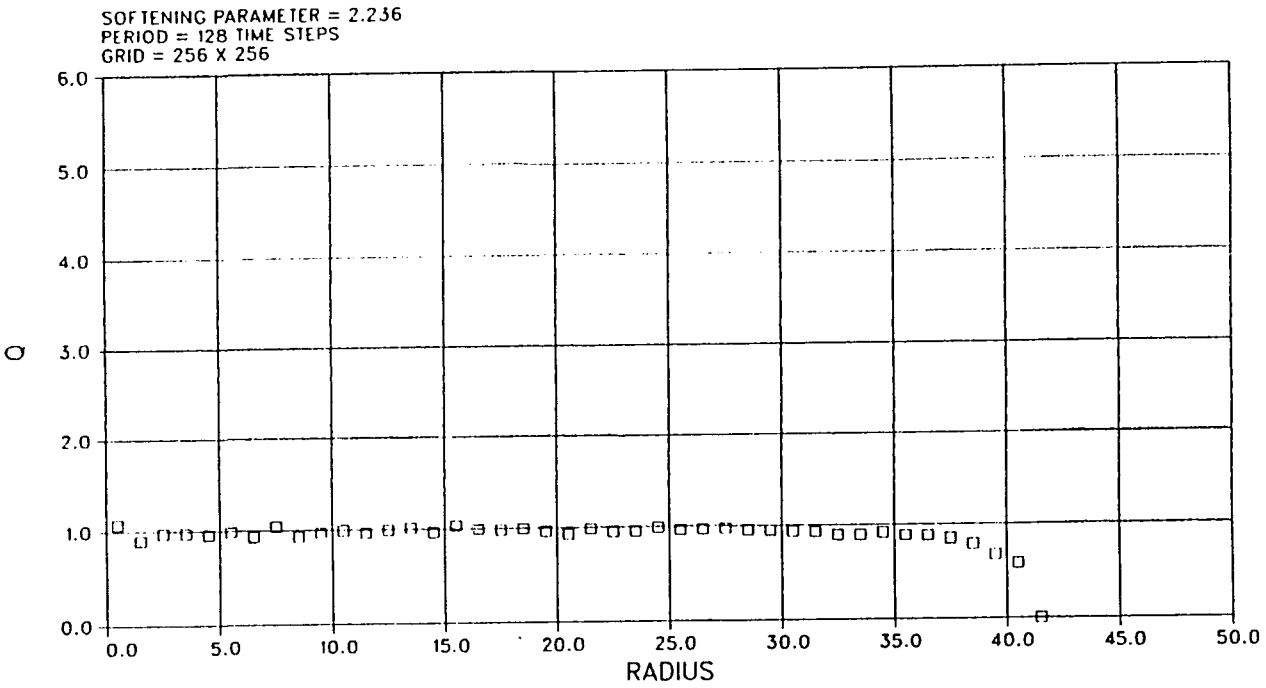

Fig 6-51: Q values for the $\Omega = 0.8\Omega_0$ Kalnajs disk for period = 0. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6-52
Q: PERIOD = 1: OMEGA = 0.8

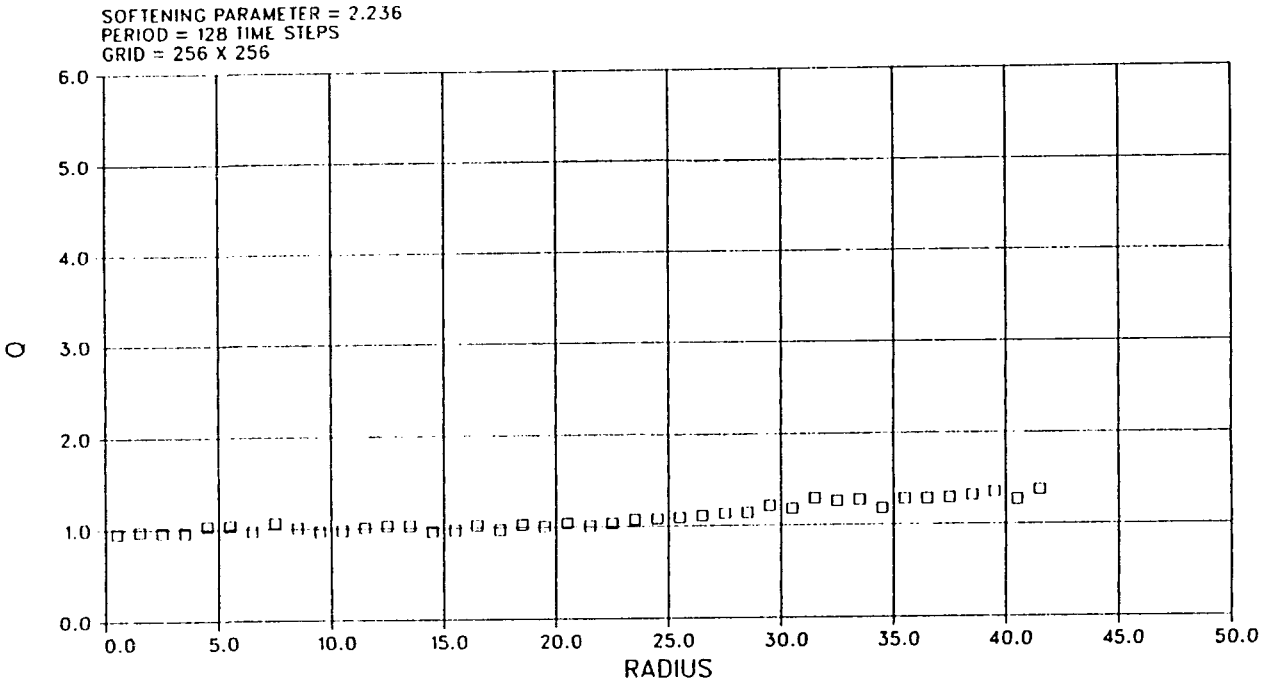

Fig 6-52: Q values for the $\Omega = 0.8\Omega_0$ Kalnajs disk for period = 1. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–53
Q: PERIOD = 2: OMEGA = 0.8

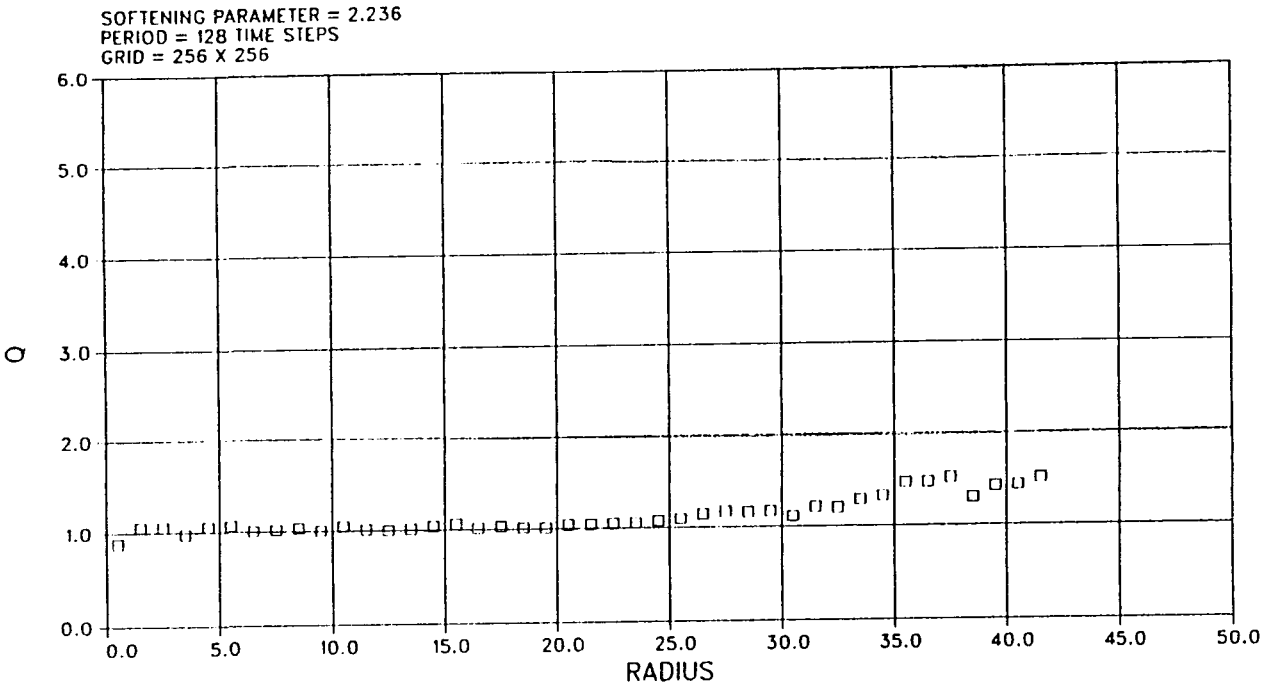

Fig 6-53: Q values for the $\Omega = 0.8\Omega_0$ Kalnajs disk for period = 2. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–54
Q: PERIOD = 3: OMEGA = 0.8

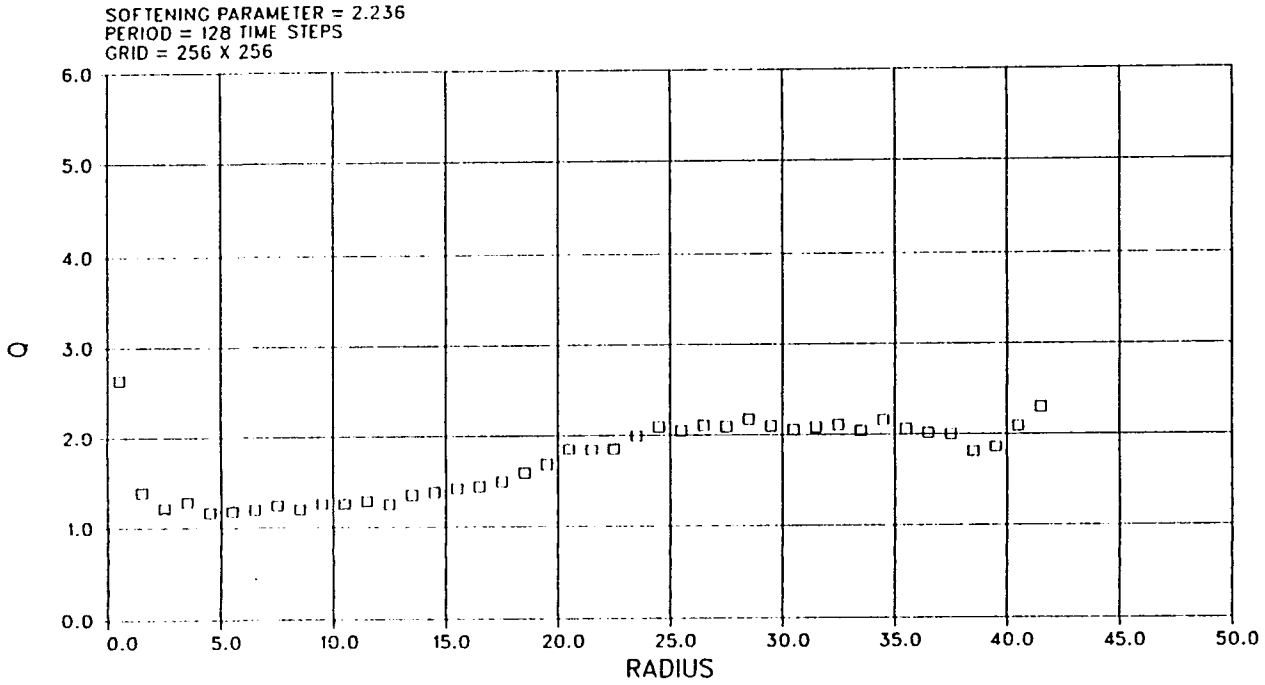

Fig 6-54: Q values for the $\Omega = 0.8\Omega_0$ Kalnajs disk for period = 3. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–55
Q: PERIOD = 4: OMEGA = 0.8

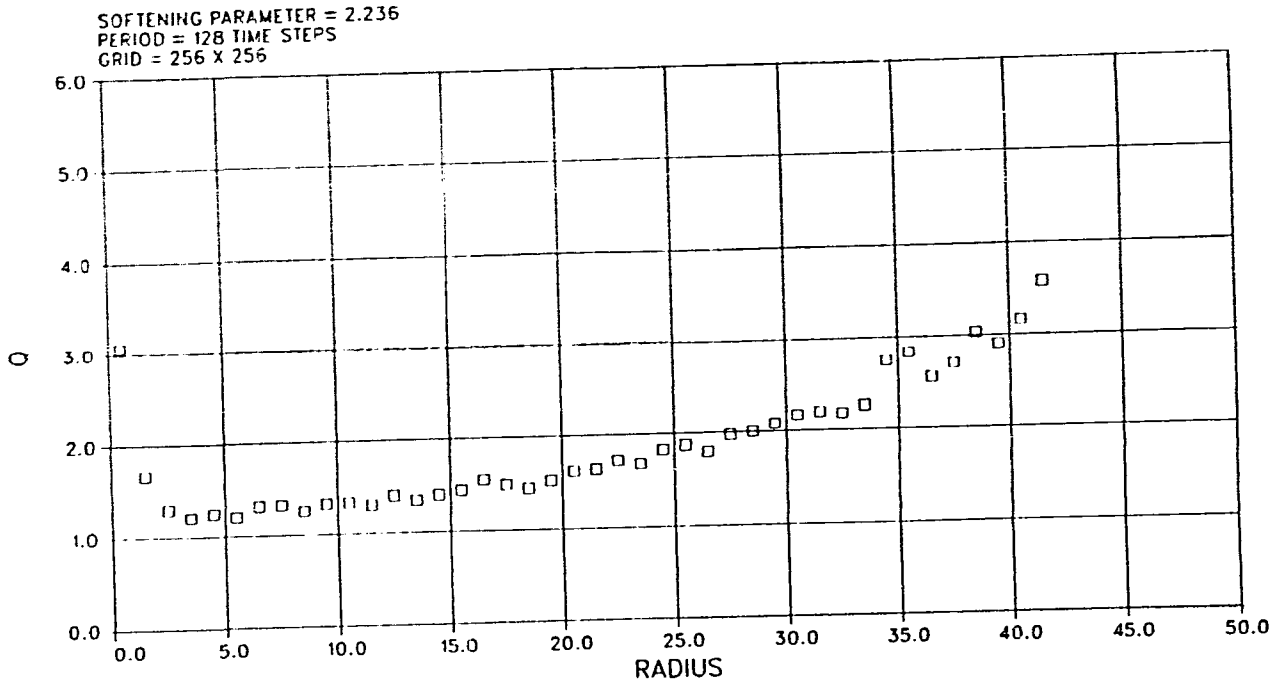

Fig 6-55: Q values for the $\Omega = 0.8\Omega_0$ Kalnajs disk for period = 4. One period is 128.8 time steps, $\epsilon^2 = 5$, on a 256 X 256 grid. Radial bins are one grid unit wide.

FIGURE 6–56
PERCENT PARTICLE SPILLS: OMEGA = 0.8

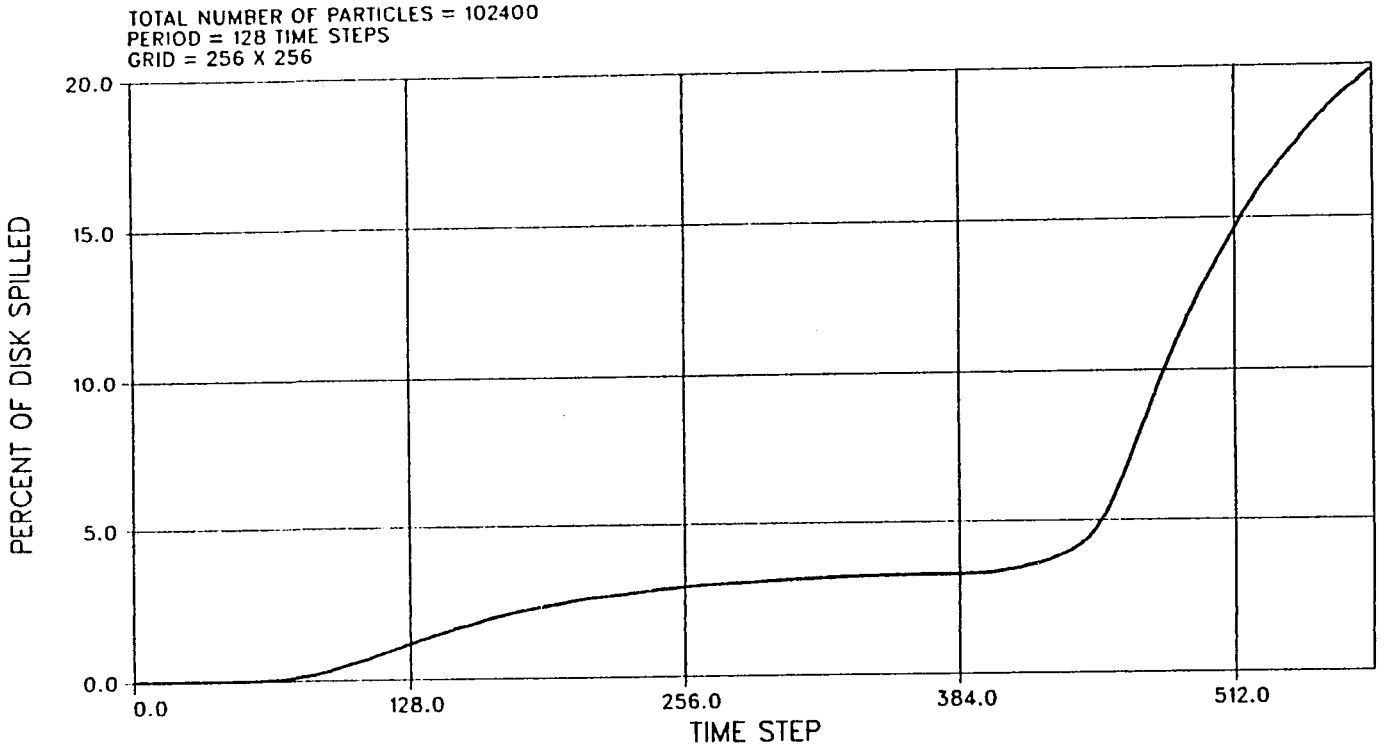

Fig 6-56: Percent of the total disk that has spilled as a function of time step. Each period is 128.8 time steps.

APPENDIX A

The Kalnajs Disk Random Loader

```
C+------------------------------------------------------------------+
C| PROGRAM: KALNAJS_LOAD.FOR, 4-AUG-90, Mark Mattox                 |
C| PURPOSE: Random load of a Kalnajs disk for the 2-D cart. grid code |
C+------------------------------------------------------------------+
      IMPLICIT NONE

      INCLUDE 'griddef.inc'     !defines grid size ngrid**2

      REAL PARTDATA(NPART,6), RUNDATA1(64), ASQ, WON, VMAX, MIDPT
      REAL RMIN, RMAX, PI, ZERO, CONST, WRK, SIGMA0, RAN
      REAL RADIUS, OMEGA0, OMEGAP, OMEGA, R, AZ, VR, VAZ
      REAL X, Y, VX, VY, VMAXR, VRDISP, VAZDISP, TVAZ, TVR
      REAL GASRAN, DISP, VEC, PHASE

      INTEGER RUNDATA2(64), STEP, STEPLIMIT, PERIOD, PART, PACTIV
      INTEGER ISEED, IOUTSPIL, NLOOP, I, J, KK

      EXTERNAL RAN, GASRAN

      COMMON /PDATA/ PARTDATA
      COMMON /RUNDATA/ RUNDATA1, RUNDATA2

      EQUIVALENCE (RUNDATA1(62),VMAX)
      EQUIVALENCE (RUNDATA1(63),ASQ)
      EQUIVALENCE (RUNDATA1(64),WON)

      EQUIVALENCE (RUNDATA2(1),STEP)
      EQUIVALENCE (RUNDATA2(2),PART)
      EQUIVALENCE (RUNDATA2(3),PACTIV)
      EQUIVALENCE (RUNDATA2(8),STEPLIMIT)

      OPEN(UNIT=3, FILE='2body.lod', STATUS='NEW', FORM='UNFORMATTED')

      DO 1 KK = 1, 64
         RUNDATA1(KK) = 0.
         RUNDATA2(KK) = 0
1     CONTINUE

C inputs
      MIDPT = REAL(NGRID/2) + 0.5

      OMEGA = 0.8
      RMIN = 0.0
      RMAX = 41.0
      VMAXR = 2.00
      VMAX = 5000.0
      ISEED = 1230

      PART = 102400
      PACTIV = PART
      STEPLIMIT = 768
      ASQ = 5.00

C constants

      PI = 4.0*ATAN(1.0)
      ZERO = 0.0E+00
      IOUTSPIL = 0

C+------------------------------------------------------------------+
```

```
C|  Grav. const.                                                          |
C|  Defined for a Kalnajs disk such that VMAXR = 1 at Rmax.               |
C+-------------------------------------------------------------------------+
      OMEGA0 = VMAXR/RMAX
      WON = (4.0*(OMEGA0**2)*(RMAX**3))/(3.0*PI*REAL(PART))
      OMEGAP = OMEGA * OMEGA0

      WRITE(6,*) 'OMEGA0  = ', OMEGA0
      WRITE(6,*) 'WON     = ', WON

      WRITE(3) RUNDATA1
      WRITE(3) RUNDATA2

      WRK = 2.0/3.0
      NLOOP = PART / NPART
      DO 5,  J = 1, NLOOP
         DO 10, I = 1, NPART
C+-------------------------------------------------------------------------+
C|  Random Kalnajs disk in R and AZ.  Spatial load.                        |
C+-------------------------------------------------------------------------+

C  Random Radial Load

900           R = RMAX * SQRT(1.0 - (1.0 - RAN(ISEED))**WRK)
              IF( (R .LT. RMIN) .OR. (R .GT. RMAX) )THEN
                 IOUTSPIL = IOUTSPIL + 1
                 GOTO 900
              END IF

C Random Azmuthal Load

              AZ = 2.0*PI*RAN(ISEED)

C Circular velocity for a Kalnajs disk

              TVAZ = R * OMEGAP
              TVR = 0.0E+00

              VRDISP = SQRT( (OMEGA0**2 - OMEGAP**2)*
     &                  (RMAX**2 - R**2)/3.0 )
              VAZDISP = VRDISP

              VR = GASRAN(ISEED,VRDISP,TVR)
              VAZ = GASRAN(ISEED,VAZDISP,TVAZ)

C Convert to Cartesian

              X  = R * COS(AZ)
              Y  = R * SIN(AZ)
              VX = VR*COS(AZ) - VAZ*SIN(AZ)
              VY = VR*SIN(AZ) + VAZ*COS(AZ)

              PARTDATA(I,1) = MIDPT + X
              PARTDATA(I,2) = MIDPT + Y
              PARTDATA(I,3) = VX
              PARTDATA(I,4) = VY
              PARTDATA(I,5) = 0.0E+00
              PARTDATA(I,6) = 0.0E+00

10         CONTINUE
```

```
          WRITE(3) PARTDATA

5       CONTINUE

        CLOSE(3)
        WRITE(6,*) ' outspills = ', IOUTSPIL

        END

        FUNCTION GASRAN(ISEED,SIGMA,AVERAG)
C+------------------------------------------------------------------------+
C|  PROGRAM:  GASRAN(ISEED,SIGMA,AVERAG)        3-JUN-1989                |
C|  PURPOSE:  To output a random Gaussian value given the dispersion and  |
C|            the mean.                                                  |
C|  INPUTS:   ISEED                Seed for random number generator      |
C|            SIGMA                Dispersion                            |
C|            AVERAG               Average                               |
C|  CALLS:    RAN(ISEED)           Random number generator               |
C+------------------------------------------------------------------------+
        REAL GASRAN, RAN, SIGMA, AVERAG, PI, W, X1, X2, Y1, Y2
        INTEGER ISET, ISEED
        DATA ISET /0/

        PI = 4.0*ATAN(1.0)
        W = 2.0*PI
        X1 = RAN(ISEED)
        X2 = RAN(ISEED)
        IF (ISET .EQ. 0) THEN
            GASRAN = SIGMA*COS(W*X2)*SQRT(-2.0*LOG(X1)) + AVERAG
            ISET = 1
        ELSE
            GASRAN = SIGMA*SIN(W*X2)*SQRT(-2.0*LOG(X1)) + AVERAG
            ISET = 0
        END IF

        RETURN
        END

        FUNCTION RAN(ISEED)
C+------------------------------------------------------------------------+
C|  PROGRAM:  RAN(ISEED)  6-JAN-1990, Mark Mattox                         |
C|  PURPOSE:  Cray Y-MP usable random number generator.                   |
C|  INPUTS:   ISEED          Seed for random number generator             |
C|  OUTPUTS:  RAN            Uniform random number between 0. and 1.      |
C+------------------------------------------------------------------------+
        REAL RAN, RTMP(64)
        INTEGER ISEED, I, J, IFF
        DATA IFF /0/

        IF (IFF .EQ. 0) THEN
            IFF = 1
            DO 5, J = 1, ISEED
               DO 10, I = 1, 64
                  RTMP(I) = RANF()
10             CONTINUE
5           CONTINUE
        END IF
        DO 15, I = 1, 64
           RTMP(I) = RANF()
```

```
15      CONTINUE
        RAN = RTMP(1)
        RETURN
        END
```

APPENDIX B

CART2D

```
      PROGRAM MAIN

      INCLUDE 'griddef.inc'      !defines grid size ngrid**2

      INTEGER NPLUS1, NPLUS2, NSCRATCH, NCOMMON
      PARAMETER (NPLUS1=NGRID+1, NPLUS2=NGRID+2)
      PARAMETER (NCOMMON=2*NGRID**2+NPLUS1**2)
      PARAMETER (NSCRATCH=NCOMMON-NGRID**2-NPLUS2**2)

      REAL RHO(NGRID,NGRID), POT(NPLUS2,NPLUS2), PARTDATA(NPART,6)
      REAL CORE(NPLUS2,NPLUS2)
      REAL SCRATCH(NSCRATCH), RUNDATA1(64), ASQ, WON, VMAX
*  comparison check array
      REAL POTCHEK(NPLUS2,NPLUS2), POTWRITE(9,9)

      INTEGER RUNDATA2(64), STEP, STEPLIMIT, KPROP(NPART,6)
      INTEGER PART, PACTIV, P

      COMMON RHO, POT, SCRATCH
      COMMON /RUNDATA/ RUNDATA1, RUNDATA2
      COMMON /PDATA/ PARTDATA
      COMMON /CENTR/ CORE
      COMMON /MISC/ INC

      EQUIVALENCE (RUNDATA1(62),VMAX)
      EQUIVALENCE (RUNDATA1(63),ASQ)
      EQUIVALENCE (RUNDATA1(64),WON)

      EQUIVALENCE (RUNDATA2(1),STEP)
      EQUIVALENCE (RUNDATA2(2),PART)
      EQUIVALENCE (RUNDATA2(3),PACTIV)
      EQUIVALENCE (RUNDATA2(8),STEPLIMIT)

*  Initial Load data

      OPEN(3,FILE='2body.lod',STATUS='OLD',FORM='UNFORMATTED')
      OPEN(4,FILE='2body.out',STATUS='NEW')

*  Scratch files

      OPEN(10,STATUS='SCRATCH',FORM='UNFORMATTED')
      OPEN(11,STATUS='SCRATCH',FORM='UNFORMATTED')
      OPEN(12,STATUS='SCRATCH',FORM='UNFORMATTED')
      OPEN(20,STATUS='SCRATCH',FORM='UNFORMATTED')
      OPEN(21,STATUS='SCRATCH',FORM='UNFORMATTED')
      OPEN(31,STATUS='SCRATCH',FORM='UNFORMATTED')
      OPEN(32,STATUS='SCRATCH',FORM='UNFORMATTED')

*  obtain initial particle positions, velocities, and attributes;
*  obtain rundata values specified in loader,
*  and initialize RHO array

       CALL GETLOAD

*  Begin stepping through time

      ZERO = 0.0E+00
      ONE = 1.0E+00

      DO 1 I = 1, STEPLIMIT
```

```
        STEP = I

C***         WRITE(6,*) ' time = ', STEP

*  The subroutine POTENT uses the density array RHO to calculate the
*  potential array.  The potential is returned in array POT, which is
*  dimensioned two larger in each dimension than RHO to allow for
*  force interpolation on grid boundaries.

        INC = 64
        NLOOP = PART/NPART
        WRITE(4,*) step, npart
****        WRITE(4,*) step, nloop*(1+(npart/inc))

          CALL POTENT

          CALL LEAPFROG

C             WRITE(4,*) step, npart
C             DO 2357, JJ = 1, NPART
C                WRITE(4,100) (PARTDATA(JJ,J), J=1,4)
C2357         CONTINUE

1     CONTINUE

100   FORMAT(1X, 4(3X, F10.4))

      END

*************************************************************************
*                        THIS IS A NEW SUBROUTINE
*************************************************************************

      SUBROUTINE POTENT

*     POTENTIAL SOLVER.  CHANGE TO BUFFER IN/OUT, 1 DEC 1981.

      INCLUDE 'griddef.inc'     !defines grid size ngrid**2

      INTEGER NPLUS1, NPLUS2, NCOMMON, NDOUBLE, NMINUS1, N3
      PARAMETER (NPLUS1=NGRID+1, NPLUS2=NGRID+2)
      PARAMETER (NMINUS1=NGRID-1, NDOUBLE=2*NGRID)
      REAL LOW(NGRID,NGRID), UP(NGRID,NGRID), HOLD(NPLUS1,NPLUS1)
      COMMON LOW,UP,HOLD
      REAL RHO(NGRID,NGRID), POT(NPLUS2,NPLUS2)
      REAL CORE(NPLUS2,NPLUS2)
      COMMON /CENTR/ CORE
      EQUIVALENCE (LOW(1,1),RHO(1,1)), (UP(1,1),POT(1,1))

      BUFFER IN (10,0) (HOLD(1,1), HOLD(NPLUS1,NPLUS1))

*     START CALCULATION.

      CALL CLEAR(UP)
      CALL FWDFT(LOW)      !transform on y direction

      CALL TRANSP          !TRANSPOSE both lower and upper arrays
      CALL SAVE(UP,1)      !store upper part of transform
```

```
      CALL CLEAR(UP)
      CALL FWDFT(LOW)      !transform on x direction on lower array

      M = LENGTH(10)
c     read(10) hold
      REWIND 10

*  convolution multiply on lower transformed array

      N3 = NDOUBLE + 2
      DO 2 M = 1, NDOUBLE
         MR = MIN0(M,N3 - M)
         DO 1 L = 1, NGRID
            RHO(L,M) = RHO(L,M)*HOLD(L,MR)
1        CONTINUE
2     CONTINUE

*  perform inverse transform on convolved function in x-direction

      CALL REVFT(LOW)          !reverse transform
      CALL SAVE(LOW,2)         !store

      CALL RESTOR(LOW,1)       !retrieve lower array, entry point in SAVE
      CALL CLEAR(UP)
      CALL FWDFT(LOW)          !transform on x-direction on upper array

*  convolution multiply on upper transformed array

      DO 4 M = 1, NDOUBLE
         MR = MIN0(M,N3 - M)
         DO 3 L = 1, NGRID
            LR = NPLUS2 - L
            RHO(L,M) = RHO(L,M)*HOLD(LR,MR)      !determine indexing on ho
3        CONTINUE
4     CONTINUE

*  perform inverse transform on convolved function in x-direction

      CALL REVFT(LOW)

      CALL COPY                !copy LOW into UP

      CALL RESTOR(LOW,2)       !bring LOW back into memory
      CALL TRANSP

      CALL REVFT(LOW)

*   LOAD POT ARRAY

      DO 6 M = 1, NGRID
         M1 = M + 1
         DO 5 L = 1, NGRID
            L1 = L + 1
            POT(M1,L1) = RHO(M,L)
5        CONTINUE
6     CONTINUE

*   ADD CENTRAL CORE POTENTIAL

      DO 10, I = 1, NPLUS2
```

```
         DO 20, J = 1, NPLUS2
            POT(I,J) = POT(I,J) + CORE(I,J)
20       CONTINUE
10     CONTINUE

*    BORDERING

       DO 7 M = 2, NPLUS1
          POT(1,M) = 2.* POT(2,M) - POT(3,M)
          POT(NPLUS2,M) = 2.* POT(NPLUS1,M) - POT(NGRID,M)
7      CONTINUE

       DO 8 M = 1, NPLUS2
          POT(M,1) = 2.* POT(M,2) - POT(M,3)
          POT(M,NPLUS2) = 2.* POT(M,NPLUS1) - POT(M,NGRID)
8      CONTINUE

       RETURN
       END

*************************************************************************
*                          THIS IS A NEW SUBROUTINE
*************************************************************************

       SUBROUTINE FWDFT(A)

       INCLUDE 'griddef.inc'      !defines grid size ngrid**2

       INTEGER NDOUBLE
       PARAMETER (NDOUBLE=2*NGRID)
       REAL A(NGRID, NDOUBLE)

       INTEGER DIR(NGRID),REV(NGRID), NUMDIGREV
       COMMON/DIGREV/DIR, REV, NUMDIGREV

*      BIT - REVERSED TRANSPOSITION

       DO 2 K = 1, NUMDIGREV
          ID = DIR(K)
          IR = REV(K)
          DO 1 KK = 1, NGRID
             TEMP = A(KK,ID)
             A(KK,ID) = A(KK,IR)
             A(KK,IR) = TEMP
1         CONTINUE
2      CONTINUE

       M = 1

       X = FLOAT(NDOUBLE)
       NLOOP = NINT(LOG(X)/LOG(2.))
*    CALL RADIX 2 REAL KERNEL
       DO 3 JJ = 1, NLOOP
          CALL R2RFTK(A,M,1)
          M = M*2
3      CONTINUE

       RETURN
       END
```

```
*************************************************************************
*                        THIS IS A NEW SUBROUTINE
*************************************************************************

      SUBROUTINE REVFT(A)

      INCLUDE 'griddef.inc'      !defines grid size ngrid**2

      INTEGER NDOUBLE, NPLUS2
      PARAMETER (NDOUBLE=2*NGRID, NPLUS2=NGRID+2)
      REAL A(NGRID,NDOUBLE)

      INTEGER DIR(NGRID),REV(NGRID), NUMDIGREV
      COMMON/DIGREV/DIR, REV, NUMDIGREV

*    COMPLEX CONJUGATE DATA

      DO 2 J = NPLUS2, NDOUBLE
         DO 1 KK = 1, NGRID
            A(KK,J) =  - A(KK,J)
1        CONTINUE
2     CONTINUE

      M = NDOUBLE

      X = FLOAT(NDOUBLE)
      NLOOP = NINT(LOG(X)/LOG(2.))
*    CALL RADIX 2 HERMITE KERNEL
      DO 3 JJ = 1, NLOOP
         M = M/2
         CALL R2HFTK(A,M,1)
3     CONTINUE

*    BIT - REVERSED TRANSPOSITION

      DO 5 K = 1, NUMDIGREV
         ID = DIR(K)
         IR = REV(K)
         DO 4 KK = 1, NGRID
            TEMP = A(KK,ID)
            A(KK,ID) = A(KK,IR)
            A(KK,IR) = TEMP
4        CONTINUE
5     CONTINUE

      RETURN
      END

*************************************************************************
*                        THIS IS A NEW SUBROUTINE
*************************************************************************

      SUBROUTINE R2 RFTK (A,M,L0)

*    RADIX 2 REAL KERNEL, FROM SANDE'S ROUTINES.

      INCLUDE 'griddef.inc'      !defines grid size ngrid**2

      INTEGER NDOUBLE
      PARAMETER (NDOUBLE=2*NGRID)
```

```
      REAL A(NGRID,NDOUBLE)
      INTEGER M,L0

      INTEGER J,J0,J1,K,K0,K1,KK,L1,M OVER 2,M2
      INTEGER K00,K01,K10,K11
      REAL ANGLE,C,IS,IU,I1,RS,RU,R1,S,TWOPI
      REAL T00,T01,T10,T11

      DATA TWOPI /6.2831 85307 17958 64769/

      L1 = L0 + M
      M OVER 2 = (M - 1)/2
      M2 = M*2

      DO 2 K = 1, NDOUBLE, M2
         K0 = K + L0 - 1
         K1 = K + L1 - 1
         DO 1 KK = 1, NGRID
            RS = A(KK,K0)
            RU = A(KK,K1)
            A(KK,K0) = RS + RU
            A(KK,K1) = RS - RU
1        CONTINUE
2     CONTINUE

      IF (M OVER 2.LT.1) GO TO 400

      DO 5 J = 1, M OVER 2
         J0 = J + 1
         J1 = M - J*2
         ANGLE = TWOPI*FLOAT(J)/FLOAT(M2)
         C = COS(ANGLE)
         S = SIN(ANGLE)

         DO 4 K0 = J0, NDOUBLE, M2
            K1 = K0 + J1
            K10 = L1 + K0 - 1
            K11 = L1 + K1 - 1
            K00 = L0 + K0 - 1
            K01 = L0 + K1 - 1

            DO 3 KK = 1, NGRID
               T00 = A(KK,K00)
               T01 = A(KK,K01)
               T10 = A(KK,K10)
               T11 = A(KK,K11)
               R1 = T10*C + T11*S
               I1 = T11*C - T10*S
               RS = T00 + R1
               IS = T01 + I1
               RU = T00 - R1
               IU = T01 - I1
               A(KK,K00) = RS
               A(KK,K01) = RU
               A(KK,K11) = IS
               A(KK,K10) =  - IU

3           CONTINUE     !KK LOOP
4        CONTINUE        !K0 LOOP
5     CONTINUE           ! J LOOP
```

```
  400 CONTINUE

      IF (M.NE.(M/2)*2) GO TO 600
      J = M/2 + 1
      DO 7 K = J, NDOUBLE, M2
         K1 = L1 + K - 1
         DO 6 KK = 1, NGRID
            A(KK,K1) =  - A(KK,K1)
6        CONTINUE
7     CONTINUE

  600 CONTINUE

      RETURN
      END

*****************************************************************************
*                          THIS IS A NEW SUBROUTINE
*****************************************************************************

      SUBROUTINE R2 HFTK (A,M,L0)

*   RADIX 2 HERMITE KERNEL, FROM SANDE'S ROUTINES.

      INCLUDE 'griddef.inc'      !defines grid size ngrid**2

      INTEGER NDOUBLE
      PARAMETER (NDOUBLE=2*NGRID)
      REAL A(NGRID,NDOUBLE)

      INTEGER M,L0
      INTEGER J,J0,J1,K,K0,K1,KK,L1,M OVER 2,M2
      INTEGER K00,K01,K10,K11
      REAL ANGLE,C,IS,IU,RS,RU,S,TWOPI
      REAL T00,T01,T10,T11

      DATA TWOPI /6.2831 85307 17958 64769/

      L1 = L0 + M
      M OVER 2 = (M - 1)/2
      M2 = M*2

      DO 2 K = 1, NDOUBLE, M2
         K0 = K + L0 - 1
         K1 = K + L1 - 1
         DO 1 KK = 1, NGRID
            RS = A(KK,K0)
            RU = A(KK,K1)
            A(KK,K0) = RS + RU
            A(KK,K1) = RS - RU
1        CONTINUE
2     CONTINUE

      IF (M OVER 2.LT.1) GO TO 400

      DO 5 J = 1,M OVER 2
         J0 = J + 1
         J1 = M - J*2
         ANGLE = TWOPI*FLOAT(J)/FLOAT(M2)
```

```
            C = COS(ANGLE)
            S = SIN(ANGLE)

            DO 4 K0 = J0, NDOUBLE, M2
               K1 = K0 + J1
               K10 = L1 + K0 - 1
               K11 = L1 + K1 - 1
               K00 = L0 + K0 - 1
               K01 = L0 + K1 - 1

               DO 3 KK = 1, NGRID
                  T00 = A(KK,K00)
                  T01 = A(KK,K01)
                  T10 = A(KK,K10)
                  T11 = A(KK,K11)
                  RS = T00 + T01
                  IS = T11 - T10
                  RU = T00 - T01
                  IU = T11 + T10
                  A(KK,K00) = RS
                  A(KK,K01) = IS
                  A(KK,K10) = RU*C + IU*S
                  A(KK,K11) = IU*C - RU*S
3              CONTINUE
4           CONTINUE
5        CONTINUE

  400 CONTINUE

         IF (M.NE.(M/2)*2) GO TO 600

         J = M/2 + 1
         DO 7 K = J, NDOUBLE, M2
            K0 = L0 + K - 1
            K1 = L1 + K - 1
            DO 6 KK = 1, NGRID
               A(KK,K0) = 2.0*A(KK,K0)
               A(KK,K1) = 2.0*A(KK,K1)
6           CONTINUE
7        CONTINUE

  600 CONTINUE

         RETURN
         END

*************************************************************************
*                          THIS IS A NEW SUBROUTINE                     *
*************************************************************************

         SUBROUTINE PGEN (WON,ASQ)

*           FREE - BOUNDARY POTENTIAL CONVOLUTION COEFFICIENTS.
*           21 MAY 1981.

         INCLUDE 'griddef.inc'      !defines grid size ngrid**2

         INTEGER NPLUS1, NPLUS2, NSCRATCH, NCOMMON, NDOUBLE, NPOT
         PARAMETER (NPLUS1=NGRID+1, NPLUS2=NGRID+2)
         PARAMETER (NCOMMON=2*NGRID**2+NPLUS1**2, NDOUBLE=2*NGRID)
```

```
      PARAMETER (NSCRATCH=NCOMMON - NPLUS1*NDOUBLE, NPOT=NPLUS1**2)
      COMMON  RHO(NPLUS1,NDOUBLE), SCRATCH(NSCRATCH)
      REAL VPOT(NPOT)
      EQUIVALENCE(RHO,VPOT)

      COEF = WON / (2.*FLOAT(NGRID))**2

      DO 2 L = 1, NPLUS1
            YSQ = ASQ + FLOAT((L - 1)**2)
         DO 1 K = 1, NPLUS1
            XSQ = YSQ + FLOAT((K - 1)**2)
            RHO(K,L) = COEF / SQRT(XSQ)
1        CONTINUE
2     CONTINUE

      CALL HERMFT(RHO)

      WRITE(10) VPOT
      REWIND 10

      RETURN
      END

*************************************************************************
*                          THIS IS A NEW SUBROUTINE
*************************************************************************

      SUBROUTINE HERMFT(A)

      INCLUDE 'griddef.inc'     !defines grid size ngrid**2

      INTEGER NPLUS1, NPLUS2, NDOUBLE
      PARAMETER (NPLUS1=NGRID+1, NPLUS2=NGRID+2)
      PARAMETER (NDOUBLE=2*NGRID)
      REAL A(NPLUS1,NDOUBLE), X

      INTEGER DIR(NGRID),REV(NGRID), NUMDIGREV, NLOOP
      COMMON/DIGREV/DIR, REV, NUMDIGREV

      CALL DIGREVGN


*     LOOP FOR EACH OF TWO DIMENSIONS

      DO 100  KXFM = 1, 2

*     CLEAR HIGH MEMORY
      DO 2 J = NPLUS2, NDOUBLE
         DO 1 KK = 1, NPLUS1
            A(KK,J) = 0.
1        CONTINUE
2     CONTINUE

*     CALL RADIX 2 HERMITE KERNEL
      M = NDOUBLE
      X = FLOAT(NDOUBLE)
      NLOOP = NINT(LOG(X) / LOG(2.))
      DO 3 JJ = 1, NLOOP
         M = M/2
         CALL HFTK(A,M,1)
```

```
3      CONTINUE

*    BIT - REVERSED TRANSPOSITION

       DO 5 K = 1, NUMDIGREV
          ID = DIR(K)
          IR = REV(K)
          DO 4 KK = 1, NPLUS1
             TEMP = A(KK,ID)
             A(KK,ID) = A(KK,IR)
             A(KK,IR) = TEMP
4         CONTINUE
5      CONTINUE

       CALL TRANSPOZ

100    CONTINUE

       RETURN
       END

*******************************************************************************
*                            THIS IS A NEW SUBROUTINE
*******************************************************************************

       SUBROUTINE DIGREVGN

*    GENERATE DIGIT REVERSAL TABLES FOR NGRID LENGTH FFT

       INCLUDE 'griddef.inc'

       INTEGER NDOUBLE
       PARAMETER (NDOUBLE=2*NGRID)
       INTEGER DIR(NGRID),REV(NGRID), NUMDIGREV
       REAL TWOPI
       INTEGER IR0,IR1,IR2,IR3,IR4,IR5,IR6,IR7,IR8
       INTEGER IR9,IR10,IR11,IR12
       INTEGER ID0,ID1,ID2,ID3,ID4,ID5,ID6,ID7,ID8
       INTEGER ID9,ID10,ID11,ID12
       INTEGER K0,K1,K2,K3,K4,K5,K6,K7,K8,K9,K10,K11,K12
       COMMON/DIGREV/DIR, REV, NUMDIGREV

       NUMDIGREV = 0

       IF(NDOUBLE .EQ.    8) GO TO    8
       IF(NDOUBLE .EQ.   16) GO TO   16
       IF(NDOUBLE .EQ.   32) GO TO   32
       IF(NDOUBLE .EQ.   64) GO TO   64
       IF(NDOUBLE .EQ.  128) GO TO  128
       IF(NDOUBLE .EQ.  256) GO TO  256
       IF(NDOUBLE .EQ.  512) GO TO  512
       IF(NDOUBLE .EQ. 1024) GO TO 1024
       IF(NDOUBLE .EQ. 2048) GO TO 2048

 8     CONTINUE
       DO 1003 K0 = 1,2
         IR0 = (K0-1)
         ID0 =  4*(K0-1)
        DO 1002 K1 = 1,2
          IR1 =   2*(K1-1) + IR0
```

```
        ID1 =   2*(K1-1) + ID0
       DO 1001 K2 = 1,2
         IR2 =   4*(K2-1) + IR1
         ID2 =     (K2-1) + ID1

                IF (ID2.LT.IR2) THEN
                  NUMDIGREV = NUMDIGREV+1
                  DIR(NUMDIGREV) = ID2 + 1
                  REV(NUMDIGREV) = IR2 + 1
                ENDIF

1001   CONTINUE
1002  CONTINUE
1003 CONTINUE
     RETURN

16    CONTINUE
      DO 2004 K0 = 1,2
        IR0 = (K0-1)
        ID0 =  8*(K0-1)
       DO 2003 K1 = 1,2
         IR1 =   2*(K1-1) + IR0
         ID1 =   4*(K1-1) + ID0
        DO 2002 K2 = 1,2
          IR2 =   4*(K2-1) + IR1
          ID2 =   2*(K2-1) + ID1
         DO 2001 K3 = 1,2
           IR3 =   8*(K3-1) + IR2
           ID3 =     (K3-1) + ID2

                  IF (ID3.LT.IR3) THEN
                    NUMDIGREV = NUMDIGREV+1
                    DIR(NUMDIGREV) = ID3 + 1
                    REV(NUMDIGREV) = IR3 + 1
                  ENDIF

 2001        CONTINUE
 2002   CONTINUE
 2003  CONTINUE
 2004 CONTINUE
      RETURN

 32    CONTINUE
       DO 3005 K0 = 1,2
         IR0 = (K0-1)
         ID0 =  16*(K0-1)
        DO 3004 K1 = 1,2
          IR1 =   2*(K1-1) + IR0
          ID1 =   8*(K1-1) + ID0
         DO 3003 K2 = 1,2
           IR2 =   4*(K2-1) + IR1
           ID2 =   4*(K2-1) + ID1
          DO 3002 K3 = 1,2
            IR3 =   8*(K3-1) + IR2
            ID3 =   2*(K3-1) + ID2
           DO 3001 K4 = 1,2
             IR4 =  16*(K4-1) + IR3
             ID4 =     (K4-1) + ID3

                   IF (ID4.LT.IR4) THEN
```

```
                  NUMDIGREV = NUMDIGREV+1
                  DIR(NUMDIGREV) = ID4 + 1
                  REV(NUMDIGREV) = IR4 + 1
                ENDIF

3001     CONTINUE
3002    CONTINUE
3003   CONTINUE
3004  CONTINUE
3005 CONTINUE
     RETURN

64   CONTINUE
     DO 4006 K0 = 1,2
       IR0 = (K0-1)
       ID0 =  32*(K0-1)
      DO 4005 K1 = 1,2
        IR1 =   2*(K1-1) + IR0
        ID1 =  16*(K1-1) + ID0
       DO 4004 K2 = 1,2
         IR2 =   4*(K2-1) + IR1
         ID2 =   8*(K2-1) + ID1
        DO 4003 K3 = 1,2
          IR3 =   8*(K3-1) + IR2
          ID3 =   4*(K3-1) + ID2
         DO 4002 K4 = 1,2
           IR4 =  16*(K4-1) + IR3
           ID4 =   2*(K4-1) + ID3
          DO 4001 K5 = 1,2
            IR5 =  32*(K5-1) + IR4
            ID5 =     (K5-1) + ID4

                IF (ID5.LT.IR5) THEN
                  NUMDIGREV = NUMDIGREV+1
                  DIR(NUMDIGREV) = ID5 + 1
                  REV(NUMDIGREV) = IR5 + 1
                ENDIF

4001      CONTINUE
4002     CONTINUE
4003    CONTINUE
4004   CONTINUE
4005  CONTINUE
4006 CONTINUE
     RETURN

128  CONTINUE
     DO 5007 K0 = 1,2
       IR0 = (K0-1)
       ID0 =  64*(K0-1)
      DO 5006 K1 = 1,2
        IR1 =   2*(K1-1) + IR0
        ID1 =  32*(K1-1) + ID0
       DO 5005 K2 = 1,2
         IR2 =   4*(K2-1) + IR1
         ID2 =  16*(K2-1) + ID1
        DO 5004 K3 = 1,2
          IR3 =   8*(K3-1) + IR2
          ID3 =   8*(K3-1) + ID2
         DO 5003 K4 = 1,2
```

```
            IR4 =  16*(K4-1) + IR3
            ID4 =   4*(K4-1) + ID3
           DO 5002 K5 = 1,2
             IR5 =  32*(K5-1) + IR4
             ID5 =   2*(K5-1) + ID4
            DO 5001 K6 = 1,2
              IR6 =  64*(K6-1) + IR5
              ID6 =     (K6-1) + ID5

                  IF (ID6.LT.IR6) THEN
                    NUMDIGREV = NUMDIGREV+1
                    DIR(NUMDIGREV) = ID6 + 1
                    REV(NUMDIGREV) = IR6 + 1
                  ENDIF

 5001       CONTINUE
 5002      CONTINUE
 5003     CONTINUE
 5004    CONTINUE
 5005   CONTINUE
 5006  CONTINUE
 5007 CONTINUE
      RETURN

256   CONTINUE
      DO 6008 K0 = 1,2
        IR0 = (K0-1)
        ID0 = 128*(K0-1)
       DO 6007 K1 = 1,2
         IR1 =   2*(K1-1) + IR0
         ID1 =  64*(K1-1) + ID0
        DO 6006 K2 = 1,2
          IR2 =   4*(K2-1) + IR1
          ID2 =  32*(K2-1) + ID1
         DO 6005 K3 = 1,2
           IR3 =   8*(K3-1) + IR2
           ID3 =  16*(K3-1) + ID2
          DO 6004 K4 = 1,2
            IR4 =  16*(K4-1) + IR3
            ID4 =   8*(K4-1) + ID3
           DO 6003 K5 = 1,2
             IR5 =  32*(K5-1) + IR4
             ID5 =   4*(K5-1) + ID4
            DO 6002 K6 = 1,2
              IR6 =  64*(K6-1) + IR5
              ID6 =   2*(K6-1) + ID5
             DO 6001 K7 = 1,2
               IR7 = 128*(K7-1) + IR6
               ID7 =     (K7-1) + ID6

                  IF (ID7.LT.IR7) THEN
                    NUMDIGREV = NUMDIGREV+1
                    DIR(NUMDIGREV) = ID7 + 1
                    REV(NUMDIGREV) = IR7 + 1
                  ENDIF

 6001        CONTINUE
 6002       CONTINUE
 6003      CONTINUE
 6004     CONTINUE
```

```
 6005    CONTINUE
 6006   CONTINUE
 6007  CONTINUE
 6008 CONTINUE

      RETURN


 512  CONTINUE
      DO 7009 K0 = 1,2
        IR0 = (K0-1)
        ID0 = 256*(K0-1)
       DO 7008 K1 = 1,2
         IR1 =   2*(K1-1) + IR0
         ID1 = 128*(K1-1) + ID0
        DO 7007 K2 = 1,2
          IR2 =   4*(K2-1) + IR1
          ID2 =  64*(K2-1) + ID1
         DO 7006 K3 = 1,2
           IR3 =   8*(K3-1) + IR2
           ID3 =  32*(K3-1) + ID2
          DO 7005 K4 = 1,2
            IR4 =  16*(K4-1) + IR3
            ID4 =  16*(K4-1) + ID3
           DO 7004 K5 = 1,2
             IR5 =  32*(K5-1) + IR4
             ID5 =   8*(K5-1) + ID4
            DO 7003 K6 = 1,2
              IR6 =  64*(K6-1) + IR5
              ID6 =   4*(K6-1) + ID5
             DO 7002 K7 = 1,2
               IR7 = 128*(K7-1) + IR6
               ID7 =   2*(K7-1) + ID6
              DO 7001 K8 = 1,2
                IR8 = 256*(K8-1) + IR7
                ID8 =     (K8-1) + ID7

                  IF (ID8.LT.IR8) THEN
                    NUMDIGREV = NUMDIGREV+1
                    DIR(NUMDIGREV) = ID8 + 1
                    REV(NUMDIGREV) = IR8 + 1
                  ENDIF

 7001         CONTINUE
 7002        CONTINUE
 7003       CONTINUE
 7004      CONTINUE
 7005     CONTINUE
 7006    CONTINUE
 7007   CONTINUE
 7008  CONTINUE
 7009 CONTINUE
      RETURN

 1024 CONTINUE
      DO 8010 K0 = 1,2
        IR0 = (K0-1)
        ID0 = 512*(K0-1)
       DO 8009 K1 = 1,2
         IR1 =   2*(K1-1) + IR0
```

```
        ID1 = 256*(K1-1) + ID0
       DO 8008 K2 = 1,2
         IR2 =   4*(K2-1) + IR1
         ID2 = 128*(K2-1) + ID1
        DO 8007 K3 = 1,2
          IR3 =   8*(K3-1) + IR2
          ID3 =  64*(K3-1) + ID2
         DO 8006 K4 = 1,2
           IR4 =  16*(K4-1) + IR3
           ID4 =  32*(K4-1) + ID3
          DO 8005 K5 = 1,2
            IR5 =  32*(K5-1) + IR4
            ID5 =  16*(K5-1) + ID4
           DO 8004 K6 = 1,2
             IR6 =  64*(K6-1) + IR5
             ID6 =   8*(K6-1) + ID5
            DO 8003 K7 = 1,2
              IR7 = 128*(K7-1) + IR6
              ID7 =   4*(K7-1) + ID6
             DO 8002 K8 = 1,2
               IR8 = 256*(K8-1) + IR7
               ID8 =   2*(K8-1) + ID7
              DO 8001 K9 = 1,2
                IR9 = 512*(K9-1) + IR8
                ID9 =      (K9-1) + ID8

                 IF (ID9.LT.IR9) THEN
                   NUMDIGREV = NUMDIGREV+1
                   DIR(NUMDIGREV) = ID9 + 1
                   REV(NUMDIGREV) = IR9 + 1
                 ENDIF

 8001          CONTINUE
 8002         CONTINUE
 8003        CONTINUE
 8004       CONTINUE
 8005      CONTINUE
 8006     CONTINUE
 8007    CONTINUE
 8008   CONTINUE
 8009  CONTINUE
 8010 CONTINUE
      RETURN

 2048 CONTINUE
      DO 9011 K0 = 1,2
        IR0 = (K0-1)
        ID0 = 1024*(K0-1)
       DO 9010 K1 = 1,2
         IR1 =   2*(K1-1) + IR0
         ID1 = 512*(K1-1) + ID0
        DO 9009 K2 = 1,2
          IR2 =   4*(K2-1) + IR1
          ID2 = 256*(K2-1) + ID1
         DO 9008 K3 = 1,2
           IR3 =   8*(K3-1) + IR2
           ID3 = 128*(K3-1) + ID2
          DO 9007 K4 = 1,2
            IR4 =  16*(K4-1) + IR3
            ID4 =  64*(K4-1) + ID3
```

```
            DO 9006 K5 = 1,2
              IR5 =  32*(K5-1) + IR4
              ID5 =  32*(K5-1) + ID4
             DO 9005 K6 = 1,2
               IR6 =  64*(K6-1) + IR5
               ID6 =  16*(K6-1) + ID5
              DO 9004 K7 = 1,2
                IR7 = 128*(K7-1) + IR6
                ID7 =   8*(K7-1) + ID6
               DO 9003 K8 = 1,2
                 IR8 = 256*(K8-1) + IR7
                 ID8 =   4*(K8-1) + ID7
                DO 9002 K9 = 1,2
                  IR9 = 512*(K9-1) + IR8
                  ID9 =   2*(K9-1) + ID8
                 DO 9001 K10 = 1,2
                   IR10 = 1024*(K10-1) + IR9
                   ID10 =      (K10-1) + ID9

                     IF (ID10.LT.IR10) THEN
                       NUMDIGREV = NUMDIGREV+1
                       DIR(NUMDIGREV) = ID10 + 1
                       REV(NUMDIGREV) = IR10 + 1
                     ENDIF

 9001           CONTINUE
 9002          CONTINUE
 9003         CONTINUE
 9004        CONTINUE
 9005       CONTINUE
 9006      CONTINUE
 9007     CONTINUE
 9008    CONTINUE
 9009   CONTINUE
 9010  CONTINUE
 9011 CONTINUE
      RETURN

      END

*****************************************************************************
*                          THIS IS A NEW SUBROUTINE
*****************************************************************************

      SUBROUTINE HFTK (A,M,L0)

*   RADIX 2 HERMITE KERNEL, FROM SANDE'S ROUTINES.

      INCLUDE 'griddef.inc'      !defines grid size ngrid**2

      INTEGER NPLUS1, NDOUBLE
      PARAMETER (NPLUS1=NGRID+1, NDOUBLE=2*NGRID)
      REAL A(NPLUS1,NDOUBLE)
      INTEGER M,L0
      INTEGER J,J0,J1,K,K0,K1,KK,L1,M OVER 2,M2
      INTEGER K00,K01,K10,K11
      REAL ANGLE,C,IS,IU,RS,RU,S,TWOPI
      REAL T00,T01,T10,T11

      DATA TWOPI/6.2831 85307 17958 64769/
```

```
      L1 = L0 + M
      M OVER 2 = (M - 1)/2
      M2 = M*2

      DO 2 K = 1, NDOUBLE, M2
         K0 = K + L0 - 1
         K1 = K + L1 - 1
         DO 1 KK = 1, NPLUS1
            RS = A(KK,K0)
            RU = A(KK,K1)
            A(KK,K0) = RS + RU
            A(KK,K1) = RS - RU
1        CONTINUE
2     CONTINUE

      IF (M OVER 2.LT.1) GO TO 400

      DO 5 J = 1,M OVER 2
         J0 = J + 1
         J1 = M - J*2
         ANGLE = TWOPI*FLOAT(J)/FLOAT(M2)
         C = COS(ANGLE)
         S = SIN(ANGLE)

         DO 4 K0 = J0, NDOUBLE, M2
            K1 = K0 + J1
            K10 = L1 + K0 - 1
            K11 = L1 + K1 - 1
            K00 = L0 + K0 - 1
            K01 = L0 + K1 - 1
            DO 3 KK = 1, NPLUS1
               T00 = A(KK,K00)
               T01 = A(KK,K01)
               T10 = A(KK,K10)
               T11 = A(KK,K11)
               RS = T00 + T01
               IS = T11 - T10
               RU = T00 - T01
               IU = T11 + T10
               A(KK,K00) = RS
               A(KK,K01) = IS
               A(KK,K10) = RU*C + IU*S
               A(KK,K11) = IU*C - RU*S
3           CONTINUE
4        CONTINUE
5     CONTINUE

  400 CONTINUE

      IF (M.NE.(M/2)*2) GO TO 600
      J = M/2 + 1
      DO 7 K = J, NDOUBLE, M2
         K0 = L0 + K - 1
         K1 = L1 + K - 1
         DO 6 KK = 1, NPLUS1
            A(KK,K0) = 2.0*A(KK,K0)
            A(KK,K1) = 2.0*A(KK,K1)
6        CONTINUE
7     CONTINUE
```

```
  600 CONTINUE

      RETURN
      END

*************************************************************************
*                          THIS IS A NEW SUBROUTINE
*************************************************************************

      SUBROUTINE TRANSPOZ

*    CYCLIC TRANSPOSITION OF 2 - INDEX ARRAY
*    GENERALIZED 2-D VERSION TO WORK IN PGEN.  5 JUNE 1989

      INCLUDE 'griddef.inc'      !defines grid size ngrid**2

      INTEGER NPLUS1, NSCRATCH, NCOMMON, NDOUBLE
      PARAMETER (NPLUS1=NGRID+1, NDOUBLE=2*NGRID)
      PARAMETER (NCOMMON=2*NGRID**2+NPLUS1**2)
      PARAMETER (NSCRATCH=NCOMMON-NPLUS1*NDOUBLE)

      REAL A(NPLUS1,NDOUBLE)
      REAL SCRATCH(NSCRATCH)
      COMMON  A,SCRATCH

      DO 2 K = 1, NGRID
         KK = K + 1
         DO 1 L = KK, NPLUS1
            TEMP = A(L,K)
            A(L,K) = A(K,L)
            A(K,L) = TEMP
1        CONTINUE
2     CONTINUE

      RETURN
      END

*************************************************************************
*                          THIS IS A NEW SUBROUTINE
*************************************************************************

      SUBROUTINE TRANSP

*     CYCLIC TRANSPOSITION OF TWO 2 - INDEX ARRAYS
*      THE TWO ARRAYS ARE CONTIGUOUS IN STORAGE
*      IN COMMON

      INCLUDE 'griddef.inc'      !defines grid size ngrid**2

      INTEGER NPLUS1, NMINUS1, NSCRATCH, NCOMMON
      PARAMETER (NPLUS1=NGRID+1, NMINUS1=NGRID-1)
      PARAMETER (NCOMMON=2*NGRID**2+NPLUS1**2)
      PARAMETER (NSCRATCH=NCOMMON-2*NGRID**2)

      REAL A(NGRID,NGRID), B(NGRID,NGRID)
      REAL SCRATCH(NSCRATCH)
      COMMON  A,B,SCRATCH

      DO 2 K = 1, NMINUS1
```

```
          KK = K + 1
          DO 1 L = KK, NGRID
             TEMP = A(L,K)
             A(L,K) = A(K,L)
             A(K,L) = TEMP
             TEMP = B(L,K)
             B(L,K) = B(K,L)
             B(K,L) = TEMP
1         CONTINUE
2       CONTINUE

        END

*************************************************************************
*                          THIS IS A NEW SUBROUTINE
*************************************************************************

        SUBROUTINE CLEAR(A)

*    CLEAR WORKING AREA

        INCLUDE 'griddef.inc'      !defines grid size ngrid**2

        REAL A(NGRID,NGRID)

        DO 2 KK = 1, NGRID
          DO 1 K = 1, NGRID
            A(K,KK) = 0.
1         CONTINUE
2       CONTINUE

        RETURN
        END

*************************************************************************
*                           THIS IS A NEW SUBROUTINE
*************************************************************************

        SUBROUTINE COPY

*    COPY IN MEMORY

        INCLUDE 'griddef.inc'      !defines grid size ngrid**2

        INTEGER NPLUS1, NPLUS2, NSCRATCH, NCOMMON
        PARAMETER (NPLUS1=NGRID+1, NPLUS2=NGRID+2)
        PARAMETER (NCOMMON=2*NGRID**2+NPLUS1**2)
        PARAMETER (NSCRATCH=NCOMMON-2*NGRID**2)

        REAL A(NGRID,NGRID), B(NGRID,NGRID)
        REAL SCRATCH(NSCRATCH)
        COMMON  A,B,SCRATCH

        DO 2 KK = 1, NGRID
           DO 1 K = 1, NGRID
             B(K,KK) = A(K,KK)     !put array A into 'high' memory
1          CONTINUE
2       CONTINUE

        RETURN
```

```
      END
*************************************************************************
*                          THIS IS A NEW SUBROUTINE
*************************************************************************

      SUBROUTINE SAVE(A,N)

*   DISK SAVE/RESTORE, BUFFER IN/OUT VERSION, 8 DECEMBER 1981.

      INCLUDE 'griddef.inc'      !defines grid size ngrid**2

      INTEGER NSQ, NSQHALF, N3
      PARAMETER (NSQ=NGRID**2, NSQHALF=NSQ/2, N3=NSQHALF+1)
      REAL A(NSQ)
      INTEGER N,K,L

      K = N + 10
      L = N + 30
      BUFFER OUT (K,0) (A(1),A(NSQHALF))
      BUFFER OUT (L,0) (A(N3),A(NSQ))
      I = LENGTH(K)
      I = LENGTH(L)

c      WRITE(K) (A(J), J=1,NSQHALF)
c      WRITE(L) (A(J), J=N3,NSQ)

      REWIND K
      REWIND L
      RETURN

      ENTRY RESTOR(A,N)

      K = N + 10
      L = N + 30
      BUFFER IN (K,0) (A(1),A(NSQHALF))
      BUFFER IN (L,0) (A(NSQHALF+1),A(NSQ))
      I = LENGTH(K)
      I = LENGTH(L)

c      READ(K) (A(J), J=1,NSQHALF)
c      READ(L) (A(J), J=N3,NSQ)

      REWIND K
      REWIND L
      RETURN
      END

*************************************************************************
*                          THIS IS A NEW SUBROUTINE
*************************************************************************

      SUBROUTINE LEAPFROG

*     PARTICLE PUSHER, SERVO FREE-BOUNDARY VECTOR VERSION. 2 NOV 1982.
*     MODIFIED FOR 2D --  28 AUG 1989.

      INCLUDE 'griddef.inc'

      REAL HALFGRID
```

```
      INTEGER NPLUS1, NPLUS2, NSCRATCH, NCOMMON, NSQ, N2SQ, NP3
      PARAMETER (NPLUS1 = NGRID + 1, NPLUS2 = NGRID + 2)
      PARAMETER (NCOMMON = 2*NGRID**2 + NPLUS1**2)
      PARAMETER (NSQ = NGRID**2, N2SQ=NPLUS2**2, NP3 = NPART+1)
      PARAMETER (NSCRATCH=NCOMMON-NGRID**2-NPLUS2**2)
      PARAMETER (HALFGRID = NGRID/2.)
      parameter (maxbin = 64)

      REAL PARTDATA(NPART,6), XXF(NPART,2), PL(NPART,9)
      REAL FORCE(NPART,2), FAC(NPART), RHO(NSQ), POT(N2SQ)
      REAL UPHASE(NP3,4), PESUM(NP3), KESUM(NP3), SCRATCH(NSCRATCH)
      REAL RUNDATA1(64), VMAX, ASQ, WON, KE, PE, MEAN(6)
*  comparison check array
      REAL POTCHEK(NPLUS2,NPLUS2), POTWRITE(9,9)
      real area(maxbin), vel(maxbin), rbin(maxbin), midpt
      real vr(npart,maxbin), vravg(maxbin), sdev(maxbin)
      real epsq(maxbin), sigma(maxbin), sdens(maxbin)
      real QTOOM(maxbin), epic(4)
      real pll(4,9), XOFF(4), YOFF(4)
      integer jpp(4), jll(4)

      INTEGER RUNDATA2(64), STEP, PART, VSPILL, CSPILL, PACTIV
      INTEGER LPROP(NP3),KPROP(NPART,6),P
      INTEGER JXX(NPART),JYY(NPART),JZZ(NPART),INDXMP(9)
      integer mtotsp(2), dens(maxbin)

      LOGICAL MODE(NPART),CSP(NP3),VSP(NP3),L1,L2,L3,L4,LTMP(NP3)
      LOGICAL CCT(NP3),VCT(NP3)
      logical spill(npart), diag

      COMMON RHO, POT, SCRATCH
      COMMON /RUNDATA/ RUNDATA1, RUNDATA2
      COMMON /PDATA/ PARTDATA
      COMMON /MISC/ INC

      EQUIVALENCE (RUNDATA1(33),MEAN(1))
      EQUIVALENCE (RUNDATA1(58),KE)
      EQUIVALENCE (RUNDATA1(59),PE)
      EQUIVALENCE (RUNDATA1(62),VMAX)
      EQUIVALENCE (RUNDATA1(63),ASQ)
      EQUIVALENCE (RUNDATA1(64),WON)

      EQUIVALENCE (RUNDATA2(1),STEP)
      EQUIVALENCE (RUNDATA2(2),PART)
      EQUIVALENCE (RUNDATA2(3),PACTIV)
      EQUIVALENCE (RUNDATA2(4),VSPILL)
      EQUIVALENCE (RUNDATA2(5),CSPILL)

*     ENTER WITH ARRAY POT LOADED

      PACTIV = 0
      PE = 0.
      KE = 0.
      mcsp = 0
      mvsp = 0
      angm = 0
      DO 1 L = 1, 6
        MEAN(L) = 0.
1     CONTINUE
```

```
      astep = real(step)
      pi = 4.0*atan(1.0)
      rmaxx = 41.0
      bdelt = 1.0
      dx = 0.2
      period = 128.0
C***      period = 200.0
      midpt = real(ngrid/2) + 0.5
      if (abs(mod(astep,period)) .eq. 0.0E+00) then
          diag = .true.
      else
          diag = .false.
      end if

      do 122, ll = 1, maxbin
         dens(ll) = 0.0
         area(ll) = 0.0
         rbin(ll) = 0.0
         vel(ll) = 0.0
         vravg(ll) = 0.0
         sdev(ll) = 0.0
122     continue

*     SET UP INDXMP

      JK = 0
      DO 2 K = -1, 1
        DO 3 J = -1, 1
          JK = JK + 1
          INDXMP(JK) = (K*NPLUS2) + J
3       CONTINUE
2     CONTINUE

*    CLEAR ARRAY RHO

      DO 4 L = 1, NSQ
        RHO(L) = 0.
4     CONTINUE

*   LOOP TO CONTROL LOADING OF PARTICLE DATA TO PARTDATA

 200  CALL DREAD(IERR)
        IF (IERR.NE.0) GO TO 17

*     GET X,Y POSITIONS, CALCULATE JXX=INDEX FOR POTENTIAL FETCH,
*     CALCULATE XXF = OFFSET OF PARTICLE FROM CELL CENTER

      DO 5 P = 1,NPART
         Y = PARTDATA(P,2)
         J = IFIX(Y)
         JXX(P) = J + 1
         XXF(P,2) = Y - FLOAT(J) - 0.5
          X = PARTDATA(P,1)
          J = IFIX(X)
          JXX(P) = JXX(P)*NPLUS2 + J + 2
          XXF(P,1) = X - FLOAT(J) - 0.5
         MODE(P) = (KPROP(P,5) .GE. 0)
5     CONTINUE

*     NOW LOAD ARRAY PL(NPART,9)
```

```
      DO 6 KC = 1, 9
        DO 7 P = 1,NPART
         JYY(P) = JXX(P)+INDXMP(KC)
7       CONTINUE
        CALL GATHER(NPART,PL(1,KC),POT,JYY)
6     CONTINUE

*     NOW CALCULATE FORCES AND ADVANCE PARTICLE

      DO 8 P = 1,NPART
         Y3 = 3.*XXF(P,2)
         X4 = 4.*XXF(P,1)
         X8 = 8.*XXF(P,1)

         FORCE(P,1) =           (1./12.) * ((-2.+X4+Y3)*PL(P, 1)
     X           +(   -X8      )*PL(P, 2) + ( 2.+X4-Y3)*PL(P, 3)
     X           +(-2.+X4      )*PL(P, 4) + (   -X8   )*PL(P, 5)
     X           +( 2.+X4      )*PL(P, 6) + (-2.+X4-Y3)*PL(P, 7)
     X           +(   -X8      )*PL(P, 8) + ( 2.+X4+Y3)*PL(P, 9))
8     CONTINUE

      DO 9 P = 1,NPART
         X3 = 3.*XXF(P,1)
         Y4 = 4.*XXF(P,2)
         Y8 = 8.*XXF(P,2)

         FORCE(P,2) =       (1./12.) * ((-2.+X3+Y4)*PL(P, 1)
     Y           +(-2.   +Y4)*PL(P, 2) + (-2.-X3+Y4)*PL(P, 3)
     Y           +(      -Y8)*PL(P, 4) + (      -Y8)*PL(P, 5)
     Y           +(      -Y8)*PL(P, 6) + ( 2.-X3+Y4)*PL(P, 7)
     Y           +(+2.   +Y4)*PL(P, 8) + ( 2.+X3+Y4)*PL(P, 9))
9     CONTINUE

      DO 10 P = 1,NPART
         VSP(P) = .FALSE.
         CSP(P) = .FALSE.
         spill(p) = .false.
         PESUM(P) = CVMGT(PL(P,5),0.,MODE(P))
         KESUM(P) = 0.
         LPROP(P) = KPROP(P,5)
10    CONTINUE

* set previous spills
      do 212, p = 1, npart
         if(partdata(p,5) .gt. 0.0) spill(p) = .true.
212    continue

* calculate total potential energy
       do 210, p = 1, npart
          pe = pe + cvmgt(0.,pl(p,5),spill(p))
210    continue

*  UPDATE PARTICLE INFO.  VSP,CSP TRUE IF PARTICLE SPILLS
*  UPHASE CONTAINS NEW VELOCITY, POSITION EVEN IF PARTICLE SPILLS
*  KE TALLY CONTAINS CONTRIBUTION FROM PARTICLE THAT SPILLS THIS STEP

       DO 11 K = 1, 2
          KP = K + 2
          DO 11 P = 1, NPART
```

```
            T1 = PARTDATA(P,KP) + FORCE(P,K)
            VSP(P) = VSP(P) .OR. (ABS(T1).GE.VMAX)
            mvsp = mvsp + CVMGT(1,0,vsp(p))
            UPHASE(P,KP) = T1
            KESUM(P) = KESUM(P) + T1**2
            T2 = PARTDATA(P,K) + T1
            CSP(P) = CSP(P) .OR. (ABS(T2-HALFGRID).GE.HALFGRID)
            mcsp = mcsp + CVMGT(1,0,csp(p))
            UPHASE(P,K) = T2
            spill(p) = (vsp(p) .or. csp(p)) .or. spill(p)
11      CONTINUE

* Calculate angular momentum (do not include spilled particles)

        do 1110, p = 1, npart
           angm = angm + ((partdata(p,1)-midpt)*(uphase(p,2)-midpt)
     &                 - (partdata(p,2)-midpt)*(uphase(p,1)-midpt))
     &                 *CVMGT(0.,1.,spill(p))
1110    continue

* Calculate total kinetic energy (avg. on half-integral steps)
        do 1178, p = 1, npart
           vx1 = partdata(p,3)
           vy1 = partdata(p,4)
           vx2 = uphase(p,3)
           vy2 = uphase(p,4)
           ke = ke + ((vx1**2 + vy1**2 + vx2**2 + vy2**2)/4.)
     &             *cvmgt(0.,1.,spill(p))
1178    continue

        DO 12 K = 1,4
        DO 12 P = 1,NPART
C***          PARTDATA(P,K) = CVMGT(UPHASE(P,K),PARTDATA(P,K),MODE(P))
           partdata(p,k) = cvmgt(partdata(p,k),uphase(p,k),spill(p))
12      CONTINUE

* Bin info for density and radial velocity dispersion

        if (diag) then

           bmax = 0.
           kbin = 1
918        if (bmax .le. rmaxx) then
           bmin = bmax
           bmax = bmin + bdelt
           rbin(kbin) = bmin + (bmax - bmin)/2.0
           do 1212, p = 1, npart
              xx = partdata(p,1) - midpt
              yy = partdata(p,2) - midpt
              vxx = partdata(p,3)
              vyy = partdata(p,4)
              az = atan2(yy,xx)
              rr = sqrt(xx**2 + yy**2)
              vmag =  sqrt(vxx**2 + vyy**2)
              vdr = vxx*xx + vyy*yy
              ct = vdr / (vmag * rr)
              vrr = vmag * ct
              if ((bmin .le. rr).and.(rr .lt. bmax))then
                dens(kbin) = dens(kbin) + 1
                area(kbin) = pi*(bmax**2 - bmin**2)
```

```
                vel(kbin) = vel(kbin) + vrr
                sdev(kbin) = sdev(kbin) + vrr**2
C***                vr(p,kbin) = vrr
              end if
1212        continue
            kbin = kbin + 1
            goto 918
            end if

C***            BUFFER OUT(45,0) (VR(1,1),VR(NPART,MAXBIN))

            klim = kbin - 1

            do 1222, k = 1, klim
               if (dens(k) .eq. 0.0) then
                   vravg(k) = 0.0
               else
                   vravg(k) = vel(k)/dens(k)
               end if
1222        continue

         end if


*     SPILLED PARTICLE MARKED BY NEGATIVE VALUE IN KPROP(P,5)
*          SET BY MASK(1) VALUE
*     VELOCITY SPILL MARKED BY 1 IN LAST BIT IN WORD
*          SET BY K
*     RETAINS OTHER ATTRIBUTES.  PARTICLE POSITION,VELOCITY CONTAIN
*          THE VALUES THAT CAUSED SPILL, IF SPILLS.

         JKL = MASK(1)
         DO 13 P = 1,NPART
            L1 = MODE(P)
            L2 = VSP(P).AND.L1
            L3 = CSP(P).AND.L1
            L4 = L1.AND.(.NOT.(L2.OR.L3))
            K = CVMGT(1,0,L2)
            CCT(P) = L3
            VCT(P) = L2
            J = LPROP(P)
            KP = (JKL+K).OR.J
            LTMP(P) = L4
            KPROP(P,5) = CVMGT(J,KP,L4)
            FAC(P) = CVMGT(1.0,0.0,L4)
13       CONTINUE

*         PE = PE+SSUM(NPART,PESUM,1)
*         KE = KE+SDOT(NPART,KESUM,1,FAC,1)
         VSPILL = VSPILL+ILSUM(NPART,VCT,1)
         CSPILL = CSPILL+ILSUM(NPART,CCT,1)
         PACTIV = PACTIV+ILSUM(NPART,LTMP,1)

         DO 14 J = 1,4
            MEAN(J) = MEAN(J)+SDOT(NPART,PARTDATA(1,J),1,FAC,1)
14       CONTINUE

         DO 15 P = 1,NPART
          JZZ(P) = (IFIX(PARTDATA(P,2)))*NGRID + IFIX(PARTDATA(P,1)) + 1
15       CONTINUE
```

```
      DO 16 P = 1,NPART
C***          KP = CVMGT(JZZ(P),1,LTMP(P))
C***          RHO(KP) = RHO(KP)+CVMGT(1.,0.,LTMP(P))
         kp = cvmgt(1,jzz(p),spill(p))
         rho(kp) = rho(kp)+cvmgt(0.,1.,spill(p))
16    CONTINUE

* update spill info in partdata
      do 22, p = 1, npart
         if(spill(p)) partdata(p,5) = 9.0
22    continue

*************************************************************************
*                          OUTPUT MOVIE FILE                            *

*       zero = 0.0E+00
*       one = 1.0E+00
*       if (diag) then
*       do 710, jj = 1, npart, inc
*         if (.not. spill(jj)) then
*         write(4,720) (partdata(jj,j), j=1,2), zero, one
*         end if
*710    continue
*       end if
720    format(1x, 4(3x, f10.4))

*************************************************************************


      CALL DWRITE(IERR)

      GO TO 200

17    CONTINUE

* Output particle info
      etot = ke - pe
      write(6,*) step, mcsp, '  total number of grid spills'
      write(6,*) step, mvsp, '  total number of velocity spills'
      write(6,*) step, angm, '  total angular momentum'
      write(6,*) step, etot, '  total energy'

      if (diag) then
C***          rewind 45
         nloop = part/npart
C***          do 1177, i = 1, nloop
C***             BUFFER IN(45,0) (VR(1,1),VR(NPART,MAXBIN))
C***             aerror = UNIT(45)
C***             do 1277, k = 1, klim
C***                do 1377, p = 1, npart
C***                   sdev(k) = sdev(k) + (vr(p,k) - vravg(k))**2
C***1377             continue
C***1277          continue
C***1177       continue

C***          rewind 45

* Calculate epic. freq. at four points in each ring
         npts = 4
```

```
      do 1657, k = 1, klim
* +X
          yp = midpt
          jp = ifix(yp)
          jpp(1) = jp + 1
          yoff(1) = yp - float(jp) - 0.5
          xp = rbin(k) + midpt
          jp = ifix(xp)
          jpp(1) = jpp(1)*nplus2 + jp + 2
          xoff(1) = xp - float(jp) - 0.5
* +Y
          yp = rbin(k) + midpt
          jp = ifix(yp)
          jpp(2) = jp + 1
          yoff(2) = yp - float(jp) - 0.5
          xp = midpt
          jp = ifix(xp)
          jpp(2) = jpp(2)*nplus2 + jp + 2
          xoff(2) = xp - float(jp) - 0.5
* -X
          yp = midpt
          jp = ifix(yp)
          jpp(3) = jp + 1
          yoff(3) = yp - float(jp) - 0.5
          xp = -rbin(k) + midpt
          jp = ifix(xp)
          jpp(3) = jpp(3)*nplus2 + jp + 2
          xoff(3) = xp - float(jp) - 0.5
* -Y
          yp = -rbin(k) + midpt
          jp = ifix(yp)
          jpp(4) = jp + 1
          yoff(4) = yp - float(jp) - 0.5
          xp = midpt
          jp = ifix(xp)
          jpp(4) = jpp(4)*nplus2 + jp + 2
          xoff(4) = xp - float(jp) - 0.5

          do 2212, kcp = 1, 9
             do 2213, np = 1, npts
               jll(np) = jpp(np) + indxmp(kcp)
2213         continue
             call gather(npts,pll(1,kcp),pot,jll)
2212      continue

* +X point
          Y3 = 3.*YOFF(1)
          X4 = 4.*XOFF(1)
          X8 = 8.*XOFF(1)
          FP =                  (1./12.) * ((-2.+X4+Y3)*PLL(1, 1)
     X       +(   -X8      )*PLL(1, 2) + ( 2.+X4-Y3)*PLL(1, 3)
     X       +(-2.+X4      )*PLL(1, 4) + (   -X8   )*PLL(1, 5)
     X       +( 2.+X4      )*PLL(1, 6) + (-2.+X4-Y3)*PLL(1, 7)
     X       +(   -X8      )*PLL(1, 8) + ( 2.+X4+Y3)*PLL(1, 9))

          XOFFN = XOFF(1) - DX
          Y3 = 3.*YOFF(1)
          X4 = 4.*XOFFN
          X8 = 8.*XOFFN
          FPN =               (1./12.) * ((-2.+X4+Y3)*PLL(1, 1)
```

```
     X          +(   -X8      )*PLL(1, 2) + ( 2.+X4-Y3)*PLL(1, 3)
     X          +(-2.+X4      )*PLL(1, 4) + (   -X8   )*PLL(1, 5)
     X          +( 2.+X4      )*PLL(1, 6) + (-2.+X4-Y3)*PLL(1, 7)
     X          +(   -X8      )*PLL(1, 8) + ( 2.+X4+Y3)*PLL(1, 9))

            XOFFP = XOFF(1) + DX
            Y3 = 3.*YOFF(1)
            X4 = 4.*XOFFP
            X8 = 8.*XOFFP
            FPP =                (1./12.) * ((-2.+X4+Y3)*PLL(1, 1)
     X          +(   -X8      )*PLL(1, 2) + ( 2.+X4-Y3)*PLL(1, 3)
     X          +(-2.+X4      )*PLL(1, 4) + (   -X8   )*PLL(1, 5)
     X          +( 2.+X4      )*PLL(1, 6) + (-2.+X4-Y3)*PLL(1, 7)
     X          +(   -X8      )*PLL(1, 8) + ( 2.+X4+Y3)*PLL(1, 9))

            FP1 = (FPP - FPN)/(2.0*DX)

            epic(1) = abs((-3./rbin(k))*fp - fp1)

* +Y point
            X3 = 3.*XOFF(2)
            Y4 = 4.*YOFF(2)
            Y8 = 8.*YOFF(2)

            FP =             (1./12.) * ((-2.+X3+Y4)*PLL(2, 1)
     Y          +(-2.    +Y4)*PLL(2, 2) + (-2.-X3+Y4)*PLL(2, 3)
     Y          +(       -Y8)*PLL(2, 4) + (       -Y8)*PLL(2, 5)
     Y          +(       -Y8)*PLL(2, 6) + ( 2.-X3+Y4)*PLL(2, 7)
     Y          +(+2.    +Y4)*PLL(2, 8) + ( 2.+X3+Y4)*PLL(2, 9))

            YOFFN = YOFF(2) - DX
            X3 = 3.*XOFF(2)
            Y4 = 4.*YOFFN
            Y8 = 8.*YOFFN

            FPN =             (1./12.) * ((-2.+X3+Y4)*PLL(2, 1)
     Y          +(-2.    +Y4)*PLL(2, 2) + (-2.-X3+Y4)*PLL(2, 3)
     Y          +(       -Y8)*PLL(2, 4) + (       -Y8)*PLL(2, 5)
     Y          +(       -Y8)*PLL(2, 6) + ( 2.-X3+Y4)*PLL(2, 7)
     Y          +(+2.    +Y4)*PLL(2, 8) + ( 2.+X3+Y4)*PLL(2, 9))

            YOFFP = YOFF(2) + DX
            X3 = 3.*XOFF(2)
            Y4 = 4.*YOFFP
            Y8 = 8.*YOFFP

            FPP =             (1./12.) * ((-2.+X3+Y4)*PLL(2, 1)
     Y          +(-2.    +Y4)*PLL(2, 2) + (-2.-X3+Y4)*PLL(2, 3)
     Y          +(       -Y8)*PLL(2, 4) + (       -Y8)*PLL(2, 5)
     Y          +(       -Y8)*PLL(2, 6) + ( 2.-X3+Y4)*PLL(2, 7)
     Y          +(+2.    +Y4)*PLL(2, 8) + ( 2.+X3+Y4)*PLL(2, 9))

            FP1 = (FPP - FPN)/(2.0*DX)

            epic(2) = abs((-3./rbin(k))*fp - fp1)

* -X point
            Y3 = 3.*YOFF(3)
            X4 = 4.*XOFF(3)
            X8 = 8.*XOFF(3)
```

```
      FP =                (1./12.) * ((-2.+X4+Y3)*PLL(3, 1)
     X   +(   -X8       )*PLL(3, 2) + ( 2.+X4-Y3)*PLL(3, 3)
     X   +(-2.+X4       )*PLL(3, 4) + (   -X8   )*PLL(3, 5)
     X   +( 2.+X4       )*PLL(3, 6) + (-2.+X4-Y3)*PLL(3, 7)
     X   +(   -X8       )*PLL(3, 8) + ( 2.+X4+Y3)*PLL(3, 9))

      XOFFN = XOFF(3) + DX
      Y3 = 3.*YOFF(3)
      X4 = 4.*XOFFN
      X8 = 8.*XOFFN
      FPN =                  (1./12.) * ((-2.+X4+Y3)*PLL(3, 1)
     X   +(   -X8       )*PLL(3, 2) + ( 2.+X4-Y3)*PLL(3, 3)
     X   +(-2.+X4       )*PLL(3, 4) + (   -X8   )*PLL(3, 5)
     X   +( 2.+X4       )*PLL(3, 6) + (-2.+X4-Y3)*PLL(3, 7)
     X   +(   -X8       )*PLL(3, 8) + ( 2.+X4+Y3)*PLL(3, 9))

      XOFFP = XOFF(3) - DX
      Y3 = 3.*YOFF(3)
      X4 = 4.*XOFFP
      X8 = 8.*XOFFP
      FPP =                  (1./12.) * ((-2.+X4+Y3)*PLL(3, 1)
     X   +(   -X8       )*PLL(3, 2) + ( 2.+X4-Y3)*PLL(3, 3)
     X   +(-2.+X4       )*PLL(3, 4) + (   -X8   )*PLL(3, 5)
     X   +( 2.+X4       )*PLL(3, 6) + (-2.+X4-Y3)*PLL(3, 7)
     X   +(   -X8       )*PLL(3, 8) + ( 2.+X4+Y3)*PLL(3, 9))

      FP1 = (FPP - FPN)/(2.0*DX)

      epic(3) = abs((-3./rbin(k))*fp - fp1)

* -Y point
      X3 = 3.*XOFF(4)
      Y4 = 4.*YOFF(4)
      Y8 = 8.*YOFF(4)

      FP =             (1./12.) * ((-2.+X3+Y4)*PLL(4, 1)
     Y   +(-2.   +Y4)*PLL(4, 2) + (-2.-X3+Y4)*PLL(4, 3)
     Y   +(      -Y8)*PLL(4, 4) + (      -Y8)*PLL(4, 5)
     Y   +(      -Y8)*PLL(4, 6) + ( 2.-X3+Y4)*PLL(4, 7)
     Y   +(+2.   +Y4)*PLL(4, 8) + ( 2.+X3+Y4)*PLL(4, 9))

      YOFFN = YOFF(4) + DX
      X3 = 3.*XOFF(4)
      Y4 = 4.*YOFFN
      Y8 = 8.*YOFFN

      FPN =              (1./12.) * ((-2.+X3+Y4)*PLL(4, 1)
     Y   +(-2.   +Y4)*PLL(4, 2) + (-2.-X3+Y4)*PLL(4, 3)
     Y   +(      -Y8)*PLL(4, 4) + (      -Y8)*PLL(4, 5)
     Y   +(      -Y8)*PLL(4, 6) + ( 2.-X3+Y4)*PLL(4, 7)
     Y   +(+2.   +Y4)*PLL(4, 8) + ( 2.+X3+Y4)*PLL(4, 9))

      YOFFP = YOFF(4) - DX
      X3 = 3.*XOFF(4)
      Y4 = 4.*YOFFP
      Y8 = 8.*YOFFP

      FPP =              (1./12.) * ((-2.+X3+Y4)*PLL(4, 1)
     Y   +(-2.   +Y4)*PLL(4, 2) + (-2.-X3+Y4)*PLL(4, 3)
     Y   +(      -Y8)*PLL(4, 4) + (      -Y8)*PLL(4, 5)
```

```
     Y          +(      -Y8)*PLL(4, 6) + ( 2.-X3+Y4)*PLL(4, 7)
     Y          +(+2.   +Y4)*PLL(4, 8) + ( 2.+X3+Y4)*PLL(4, 9))

            FP1 = (FPP - FPN)/(2.0*DX)

            epic(4) = abs((-3./rbin(k))*fp - fp1)

* avg for kbin
            epsq(k) = (epic(1)+epic(2)+epic(3)+epic(4))/4.0

*            write(6,*) 'epic. freq. four corners:', rbin(k)
*            write(6,*) epic(1)
*            write(6,*) epic(2)
*            write(6,*) epic(3)
*            write(6,*) epic(4)
*            write(6,*) ' '
1657      continue

          do 1477, k = 1, klim
             if (dens(k)-1 .ne. 0.0E+00) then
             sigma(k) = sqrt(sdev(k)/(dens(k)-1))
             end if
             if (area(k) .ne. 0.0E+00) then
             sdens(k) = dens(k)/area(k)
             end if
1477      continue

          do 1658, k = 1, klim
             if (sdens(k) .ne. 0.0E+00) then
             QTOOM(k) = (sigma(k)*sqrt(abs(epsq(k))))/
     &                  (3.36*WON*sdens(k))
             end if
1658      continue

          write(6,5577)
5577      format(7x,'R',15X,'Q',13X,'SIGMA',10X,'SURF DENS',
     &          8X,'VR AVG',12X,'N',15X,'K',7X)
          write(6,5578)
5578      format(1x,114('='))
          do 1677, k = 1, klim
             write(6,7777) rbin(k), QTOOM(k), sigma(k),
     &       sdens(k), vravg(k), dens(k), sqrt(abs(epsq(k)))
1677      continue
          write(6,5579)
5579      format(1x,114('-'))
          write(6,*) ' '
7777      format(1x, 4(E14.7, 2x), E14.7, 2x, I14,
     &          2x, E14.7, 2x)

       end if

* MEAN(J) CONTAINS MEAN POSITION,MEAN VELOCITY
* MEAN(5-6) CONTAIN ACCLERATIONS TO GALAXY WITHIN FRAME TO BRING
* GALAXY TO FRAME CENTER

       T1 = 1./FLOAT(PACTIV)
       DO 18 J = 1,4
          MEAN(J) = MEAN(J)*T1
18     CONTINUE
       DO 19 J = 1,2
```

```
         MEAN(J) = MEAN(J) - HALFGRID
19     CONTINUE
       DO 20 J = 1,2
         MEAN(J+4) = -(0.4*MEAN(J+2)+0.5*MEAN(J))
20     CONTINUE


       RETURN
       END

*************************************************************************
*                          THIS IS A NEW SUBROUTINE
*************************************************************************

       SUBROUTINE DREAD(IERR)
*
*    HANDLE DISK I/O FOR PARTICLE PUSHER, ETC.
*      Y-MP VERSION. BUFFER IN/OUT.  29 MAY 1989
*
*    ASSUME DIRECT PASS THROUGH PARTICLE DATA WHENEVER ACCESSED
*       (NO RANDOM ACCESS)
*
*    READS FROM ONE UNIT, WRITES TO OTHER.  ALTERNATES ON INTEGRATION
*       STEPS. USES UNITS 20,21
*
*    ALWAYS READS FULL PAGE OF NPART PARTICLES
*       PAGE COUNTERS RPAGE, WPAGE EQUIVALENCED TO RUNDATA2(29),(30)
*
*    NO DISK ERROR HANDLING
*    NO CHECKS ON PROPER CALLING SEQUENCES FOR DISK ACCESSES


         INCLUDE 'griddef.inc'

         INTEGER RUNDATA2(64),IERR,STEP,RPAGE,WPAGE
         INTEGER RUNIT, WUNIT, N6

         PARAMETER (N6=6*NPART)

         REAL PARTDATA(NPART,6), SPHASE(NPART,6), RUNDATA1(64)
         REAL RPHASE(N6),WPHASE(N6)

         SAVE RPHASE,WPHASE

         COMMON /RUNDATA/ RUNDATA1, RUNDATA2
         COMMON /PDATA/ PARTDATA

         EQUIVALENCE (PARTDATA(1,1),SPHASE(1,1))
         EQUIVALENCE (RUNDATA2(1),STEP)
         EQUIVALENCE (RUNDATA2(29),RPAGE)
         EQUIVALENCE (RUNDATA2(30),WPAGE)

         DATA RUNIT,WUNIT/21,20/

*        USE DATA TO FORCE STATIC STORAGE, SET TO LEGAL VALUES INITIALLY

         RUNIT=20+MOD(STEP+1,2)
         IF (RPAGE.LE.0) GO TO 10
         K=LENGTH(RUNIT)
*        CALL TO LENGTH FORCES COMPLETION OF PREVIOUS BUFFER IN
```

```
      IF (K.LE.0) GO TO 13
      CALL SCOPY(N6,RPHASE,1,SPHASE,1)
      GO TO 11
   10 READ(RUNIT,END=13) SPHASE
   11 CONTINUE
      BUFFER IN (RUNIT,0) (RPHASE(1),RPHASE(N6))
      IERR=0
      RPAGE=RPAGE+1
      RETURN

   13 IERR=1
      K=LENGTH(WUNIT)
*     CALL TO FUNCTION LENGTH FORCES COMPLETION OF BUFFER OUT WRITE
      RPAGE=0
      WPAGE=0
      REWIND 20
      REWIND 21
      RETURN

      ENTRY DWRITE(IERR)

      WUNIT=20+MOD(STEP+2,2)
      IF (WPAGE.LE.0) GO TO 20
      K=LENGTH(WUNIT)
*     LENGTH FUNCTION FORCES COMPLETION OF PREVIOUS BUFFEROUT TO WRITE
      IF (K.LE.0) GO TO 23
   20 CALL SCOPY(N6,SPHASE,1,WPHASE,1)
      BUFFER OUT (WUNIT,0) (WPHASE(1),WPHASE(N6))
   21 IERR=0
      WPAGE=WPAGE+1
      RETURN

   23 IERR=1
      RETURN

      END

*****************************************************************************
*                          THIS IS A NEW SUBROUTINE                         *
*****************************************************************************

      SUBROUTINE GETLOAD

      INCLUDE 'griddef.inc'      !defines grid size ngrid**2

      INTEGER NPLUS1, NPLUS2, NSCRATCH, NCOMMON, NSQ
      PARAMETER (NPLUS1=NGRID+1, NPLUS2=NGRID+2, NSQ = NGRID**2)
      PARAMETER (NCOMMON=2*NGRID**2+NPLUS1**2)
      PARAMETER (NSCRATCH=NCOMMON-NGRID**2-NPLUS2**2)

      REAL RHO(NSQ), POT(NPLUS2,NPLUS2), SCRATCH(NSCRATCH)
      REAL CORE(NPLUS2,NPLUS2), MIDPT
      REAL PARTDATA(NPART,6), RUNDATA1(64), ASQ, WON, VMAX

      INTEGER RUNDATA2(64), STEP, STEPLIMIT, P
      INTEGER PART, PACTIV

      COMMON RHO, POT, SCRATCH
      COMMON /RUNDATA/ RUNDATA1, RUNDATA2
      COMMON /PDATA/ PARTDATA
```

```
      COMMON /CENTR/ CORE

      EQUIVALENCE (RUNDATA1(63), ASQ)
      EQUIVALENCE (RUNDATA1(64), WON)

      EQUIVALENCE (RUNDATA2(2),PART)

      READ(3) RUNDATA1
      READ(3) RUNDATA2

*  Determine convolution coefficients with call to PGEN, called once,
*  at the beginning of program.  The coefficients are written to
*  record 10 and recalled when needed for the convolution multiplies
*  in subroutine POTENT.

      CALL PGEN(WON,ASQ)

*  LOAD CORE POTENTIAL (midpt calculated for pot grid, not rho grid)

*      CNTRM = 1000.0
      DO 10, I = 1, NPLUS2
         DO 20, J = 1, NPLUS2
         CORE(I,J) = 0.0E+00
*            MIDPT = REAL(NGRID/2) + 0.5 + 1.0
*            XSQ = (REAL(J) - 0.5 - MIDPT)**2
*            YSQ = (REAL(I) - 0.5 - MIDPT)**2
*            RSQ = XSQ + YSQ
*            IF (RSQ .EQ. 0.0E+00) THEN
*            CORE(I,J) = WON*CNTRM/0.01
*            ELSE
*            CORE(I,J) = WON*CNTRM/SQRT(RSQ)
*            END IF
20       CONTINUE
10    CONTINUE

      DO 1 L = 1, NSQ
        RHO(L) = 0.
1     CONTINUE

      NLOOP = PART / NPART
      DO 3 I = 1, NLOOP
         READ(3) PARTDATA
         WRITE(20) PARTDATA
         DO 2 P = 1,NPART
           KP = (IFIX(PARTDATA(P,2)))*NGRID + IFIX(PARTDATA(P,1)) + 1
           RHO(KP) = RHO(KP) + 1.
2        CONTINUE
3     CONTINUE

      REWIND(20)
      CLOSE(3)

      RETURN

       END
*************************************************************************
*                          THIS IS A NEW SUBROUTINE
*************************************************************************
       SUBROUTINE CINEMA
```

```
* This routine writes a movie file to unit 4.
* It is designed to buffer in all pages of particles, and
* skip over INC number of particles.

      INCLUDE 'griddef.inc'     !defines grid size ngrid**2
      REAL PARTDATA(NPART,6), RUNDATA1(64)
      INTEGER RUNDATA2(64), STEP, PART

      COMMON /PDATA/ PARTDATA
      COMMON /RUNDATA/ RUNDATA1, RUNDATA2

      EQUIVALENCE (RUNDATA2(1),STEP)
      EQUIVALENCE (RUNDATA2(2),PART)

      zero = 0.0
      one = 1.0

      INC = 40
      NLOOP = PART/NPART

      WRITE(4,*) step, nloop*(1+(npart/inc))
      DO 5, I = 1, NLOOP

         CALL DREAD(IERR)
         IF (IERR .NE. 0 ) THEN
             WRITE(6,*) 'ERROR IN CINEMA'
             GOTO 5
         END IF

         DO 10, JJ = 1, NPART, INC
            WRITE(4,100) (PARTDATA(JJ,J), J=1,2), zero, one
10       CONTINUE

5     CONTINUE

100   FORMAT(1X, 2(F7.2, '     ', F7.2))

      RETURN
      END
```